Berner Fachhochschule
Hochschule für Technik und Informatik
Software-Schule Schweiz

Diplombericht

HOVISSE Workflow-Editor

Klasse MAS-06-01, September 2008, Version 1.0

Abstract

Diese Arbeit befasst sich mit einem Workflow Editor, welcher es ermöglicht, die im Rahmen des HOVISSE Projekts verwendeten Workflows von medizinischen Operationen grafisch zu erfassen und zu bearbeiten. Des Weiteren kann der erfasste Workflow mit der bereits vorhandenen Workflow Engine simuliert und so auf dessen Korrektheit überprüft werden.

Keywords

Workflow, Java, Swing, GUI, JAXB, XML, HOVISSE, Editor

Diplomarbeit Nr. MAS-06-01.05

Student:
Edgar Baumeler
Seidenweg 73
3012 Bern
Tel. G. 031 330 13 38

Betreuer:
Beatrice Amrhein
Wankdorffeldstrasse 102
3014 Bern
Tel. G. 031 848 32 76

Stephan Fischli
Wankdorffeldstrasse 102
3014 Bern
Tel. G. 031 848 32 74

Experte:
Urs Künzler
Wankdorffeldstrasse 102
3014 Bern
Tel. G. 032 321 63 43

Inhaltsverzeichnis

1 Einleitung 5
1.1 Zweck des Dokuments 5
1.2 Ausgangslage / Aufgabenstellung 5
1.3 Leserkreis 5
1.4 Übersicht 6
1.5 Nomenklatur 6
2 Analyse 7
2.1 Der Workflow und seine Abhängigkeiten 7
2.2 User Interface 9
2.3 Zustandsdiagramm 9
2.3.1 Vorhandene Main Menu Punkte pro Zustand 10
2.3.2 Vorhandene Menu Punkte pro Tree Element Kontext Menu 10
2.4 Aktivitätsdiagramme 11
2.4.1 Programm starten 11
2.4.2 Neuer Workflow erstellen 12
2.4.3 Workflow laden 13
2.4.4 Workflow schliessen 14
2.4.5 Workflow speichern 15
2.4.6 Simulation Modus anwählen und Simulation starten 16
3 Prototypen 17
3.1 JAXB 17
3.2 Menu Handling 18
3.3 JTree 19
3.4 Panel Prototype 19
3.5 Test der Workflow Engine 20
4 Design 21
4.1 Einführung 21
4.2 Klassendiagramm (Übersicht) 22
4.3 Support Layer 23
4.3.1 Pakete 23
4.3.2 Environment Klasse 23
4.3.3 Exception Klassen 23
4.4 Data Layer 24
4.4.1 Pakete 24
4.4.2 Data Model 25
4.4.3 Presentation Model 26
4.4.4 dto workflow 27
4.4.5 dto inventory 28
4.4.6 dto catalog 29
4.4.7 dto.util 30
4.5 Business Layer 31
4.5.1 Pakete 31
4.5.2 Command Handler Klasse 31
4.5.3 Simulations Klassen 32
4.6 Presentation Layer 33

4.6.1 Pakete ... 33
4.6.2 Dialog ... 33
4.6.3 Menu und menu.action ... 34
4.6.4 View ... 34
4.7 Sequenz Diagramme ... 36
4.7.1 Starten des HOVISSE Workflow Editor ... 36
4.7.1.1 Initialisieren der Main View (Init Main View) ... 37
4.7.2 Öffnen des Workflow (Open Workflow) ... 38
4.7.2.1 Erzeugen des Baums (Create Tree) ... 40
4.7.3 Selektieren eines Elements (Select Element) ... 41
4.7.4 Hinzufügen eines Elements (Add ActionGroup) ... 41
5 Lösungsbeschreibung ... 43
5.1 Datenfluss ... 43
5.1.1 Erste Idee ... 43
5.1.2 Umgesetzter Datenfluss ... 44
5.2 Abweichungen zum Pflichtenheft ... 45
5.3 Nützliche Hinweise ... 45
5.3.1 Schliessen mit dem X Button ... 45
5.3.2 Ausfüllen eines Panels mit dem Tree Panel ... 46
6 Tools & Technologien ... 47
6.1 Entwicklungsumgebung ... 47
6.2 Weitere Tools ... 47
6.2.1 Subversion (SVN) ... 47
6.2.2 UML ... 47
6.3 Swing ... 48
6.4 JAXB ... 48
6.5 Log4j ... 48
7 Test ... 49
7.1 Usability Test ... 49
7.1.1 actor und object ... 50
7.1.2 Proposition ... 50
7.2 System Test Cases ... 52
7.3 Test Cases ... 52
8 Fazit ... 59
8.1 Vorgehen / Persönlicher Eindruck ... 59
8.2 Resultat ... 59
8.3 Ausblick ... 59
9 Benutzerhandbuch ... 60
9.1 Konfiguration Files ... 60
9.1.1 hwfe.properties ... 60
9.1.2 log4j.properties ... 60
9.2 Laden / Öffnen / Speichern eines Workflows ... 61
9.3 Selektieren / Bearbeiten von Elementen ... 61
9.4 Hinzufügen / Entfernen von Tree Elementen ... 62
9.5 Simulation Modus / Edit Modus ... 63
9.6 Simulation Menu ... 63
9.7 Simulation ... 63
A. XML Schemas & Files ... 64
B. Glossar ... 64

C. Referenzen.....65
D. Tabellenverzeichnis.....66
E. Abbildungsverzeichnis.....66
F. Codeverzeichnis.....67

1 Einleitung

1.1 Zweck des Dokuments

Dieses Dokument ist der Abschlussbericht zur Diplomarbeit (Master Thesis) des HOVISSE Workflow Editors. Es soll darin aufgezeigt werden, welche Arbeiten im Rahmen dieser Diplomarbeit durchgeführt wurden.

Es werden die Überlegungen zu den einzelnen Phasen, also Analyse, Design und Test beschrieben, sowie die aufgetretenen Probleme thematisiert und auf die daraus resultierenden Erkenntnisse eingegangen.

Zudem ist dies die eigentliche Dokumentation zum Projekt und zur weiteren Verwendung der Software. Detaillierte Informationen zu den Klassen und Methoden sind in der zugehörigen javadoc zu finden. Überdies wurden die im Code potenziell unklaren Stellen soweit dokumentiert, dass die entsprechenden Schritte nachvollzogen werden können.

1.2 Ausgangslage / Aufgabenstellung

HOVISSE ist ein Forschungsprojekt der Berner Fachhochschule auf dem Gebiet der medizinischen Virtual Reality Forschung. Dabei geht es um die Planung, Simulation und Optimierung von Workflows für medizinische Operationen. Der Rahmen dieser Arbeit bestand aus der Aufgabe, einen Editor zu erstellen, welcher die Erfassung und Bearbeitung eines solchen Workflows auf einfache Art ermöglicht. Der Workflow wird dabei in einem XML-Dokument beschrieben, welches entsprechend gelesen oder geschrieben werden soll. Mit Hilfe der vorhandenen Workflow-Engine soll der erstellte Workflow simuliert werden, um so auf seine Richtigkeit überprüft werden zu können.

Eine detaillierte Übersicht über das Projekt und seinen Kontext ist im Kapitel 2 des Pflichtenhefts [2] (Übersicht Projekt) zu finden.

1.3 Leserkreis

Dieses Dokument richtet sich in erster Linie an den Experten, die Betreuer, sowie an alle interessierten Personen, die sich mit diesem Thema auseinandersetzen, bzw. die Software verwenden oder weiterentwickeln möchten.

1.4 Übersicht

Dieses Dokument ist wie folgt organisiert:

Kapitel 1	Vermittelt einen Überblick über das Dokument, dessen Zweck, Umfang und Leserkreis, sowie die Ausgangslage.
Kapitel 2	Dokumentiert die auf dem Pflichtenheft aufbauende Analyse.
Kapitel 3	Dieses Kapitel beschreibt die erstellten Prototypen.
Kapitel 4	Beschreibt das Design, aufbauend auf die vorangehenden Kapitel.
Kapitel 5	Hier werden die eigentlichen Überlegungen zur Lösung beschrieben.
Kapitel 6	Erläutert die eingesetzten Tools und Technologien.
Kapitel 7	Geht auf die durchgeführten Tests ein.
Kapitel 8	Enthält das Fazit und die Schlussfolgerungen.
Kapitel 9	Enthält ein kurzes Benutzerhandbuch.
Anhang	Enthält das Glossar, die Referenzen, sowie Code-, Tabellen- und Abbildungsverzeichnisse.

1.5 Nomenklatur

Das Dokument ist in Deutsch gehalten, dort wo es jedoch sinnvoll erschien, wurden die englischen Originalbegriffe verwendet. Zu Gunsten einer besseren Lesbarkeit wurde für sämtliche Bezeichnungen nur die männliche Form verwendet.

2 Analyse

Ein erheblicher Teil der System Analyse, der GUI Prototypen, sowie die Beschreibung der XML Schnittstelle wurde bereits im Rahmen des Pflichtenhefts [02] erstellt und ist in demselben zu finden. Nachfolgend wird auf der Basis des Pflichtenhefts aufgebaut.

2.1 Der Workflow und seine Abhängigkeiten

Nachfolgend soll der Workflow bereits einmal als Baumsicht dargestellt werden. Ein Workflow ist aus verschiedenen Elementen zusammengesetzt. In blau sind die verschiedenen Levels von Handlungen, in grün die entsprechenden Werte für die jeweiligen Level dargestellt. Das Ganze stellt sich aber doch etwas komplexer dar, als in der nachfolgenden Grafik dargestellt wird. Diverse Elemente sind nur Referenzen auf ein Inventory File oder das Catalog File.

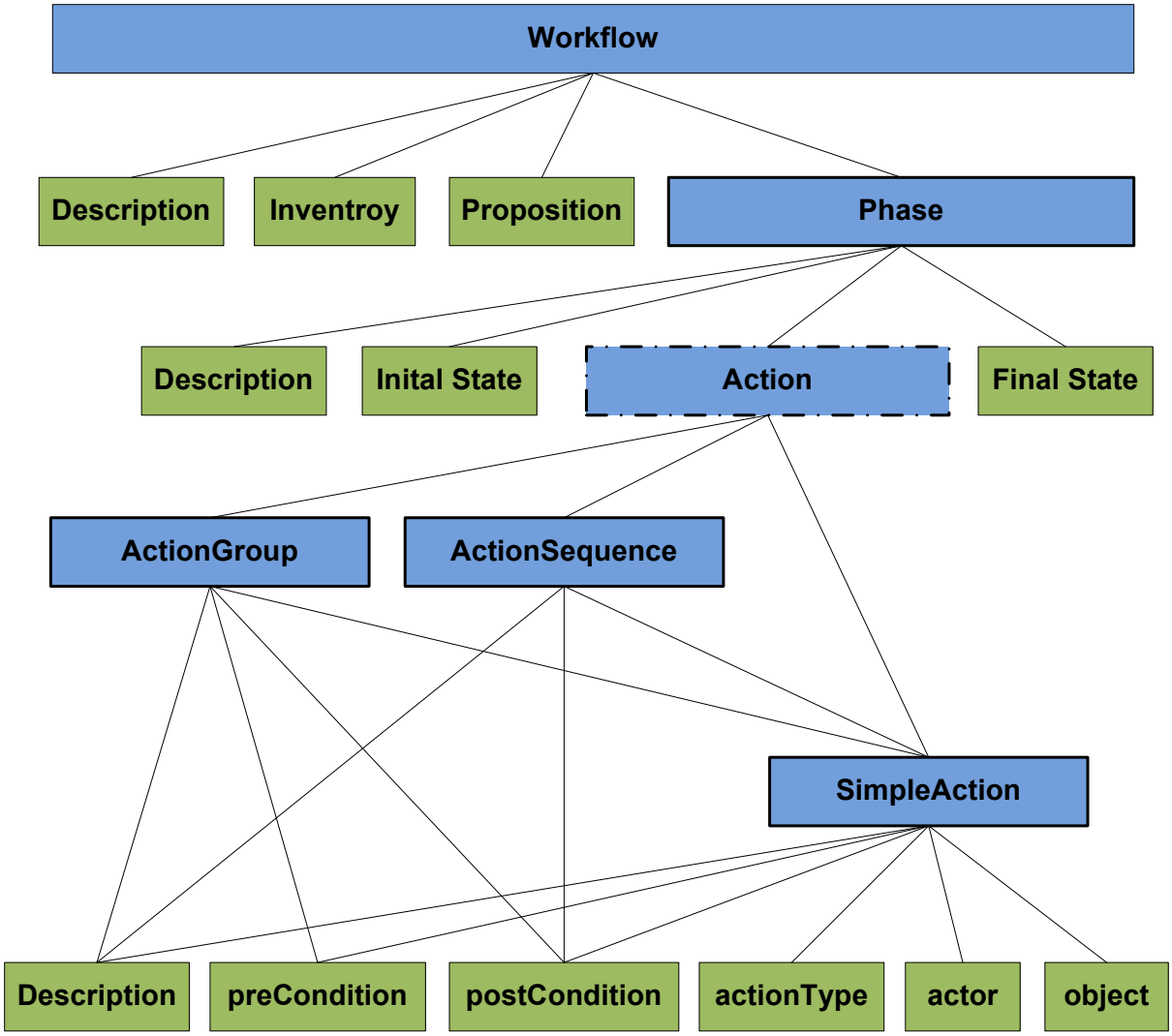

Abbildung 1: Datenstruktur des Workflows

Das Element Inventory ist, wie der Name schon vermuten lässt, eine Referenz auf das entsprechende Inventory File. Actor und object sind Referenzen auf das Inventory File (1), welches seinerseits weiter auf das Catalog File referenziert (2). Des Weiteren bestehen aber bereits Abhängigkeiten innerhalb des Workflows. So referenzieren InitialState, FinalState, preCondition und postCondition auf die Proposition des Workflow.
Ein ganz spezieller Fall sind die ActionTypes. Sie sind einerseits abhängig von der catRef, welche auf das entsprechende Element im Catalog referenziert (3). Zudem sind sie aber auch noch indirekt von den vorhandenen actor abhängig, bzw. dessen Referenz (1), (2), welche zusätzliche actionType (4) enthalten. Somit können durch die Auswahl der actor die vorhandenen actionType einer Action beeinflusst werden! Weiter ist zu erkennen, dass ein actor auf eine grosse Menge von Daten (2) referenzieren kann, die dann entsprechend dargestellt werden sollten.
Mit den nachfolgenden Ausschnitten aus den entsprechenden Files werden diese Abhängigkeiten grafisch dargestellt.

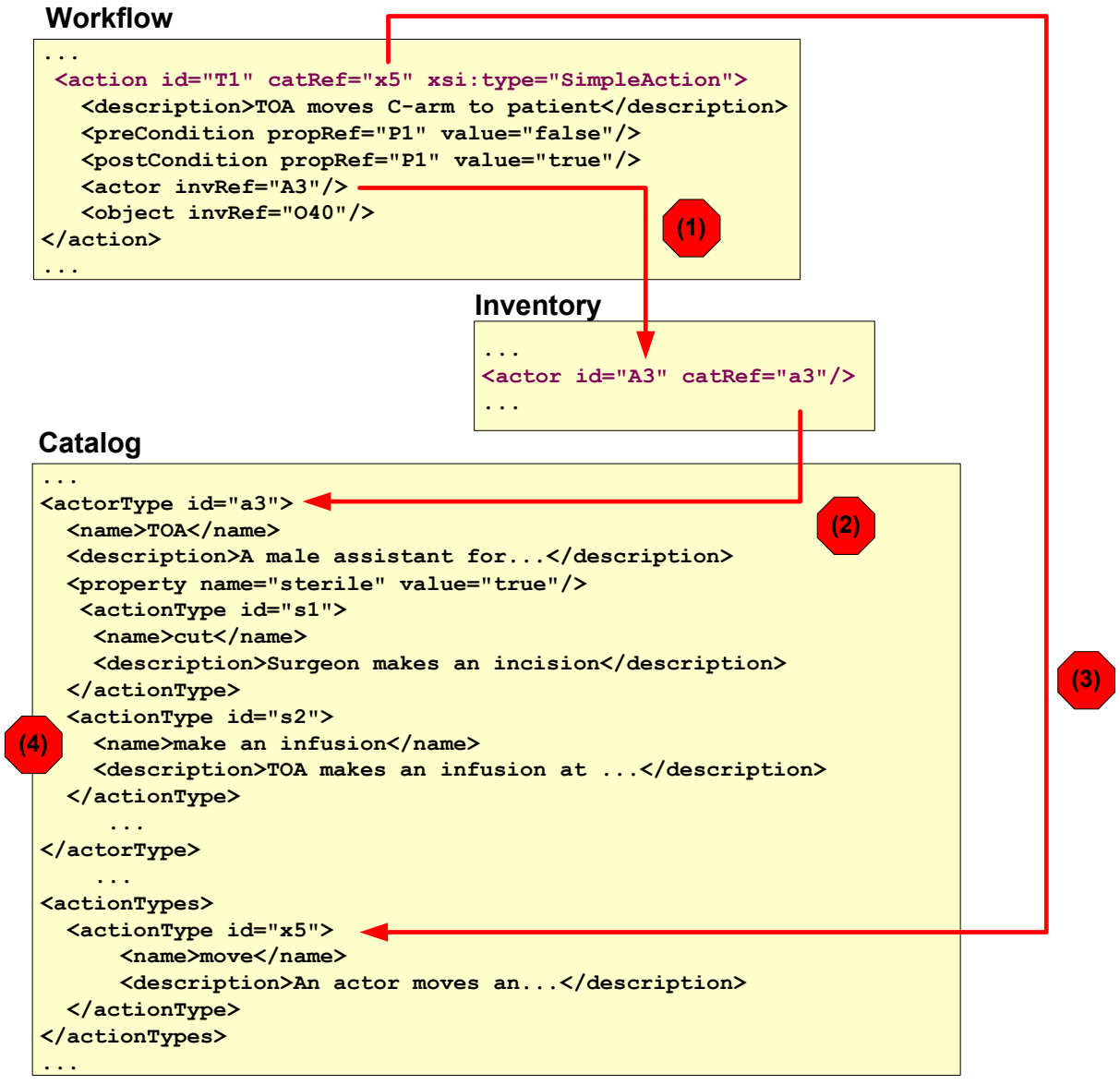

Abbildung 2: Abhängigkeiten des Workflows

2.2 User Interface

Bei der Analyse des User Interfaces wurde festgestellt, dass nebst den im Pflichtenheft beschriebenen Panels noch zusätzliche Hilfsdialoge zum Öffnen, Speichern, bzw. für Hinweise und Fehlermeldungen benötigt werden.
Zusätzlich zu den eigenen Erkenntnissen wurde bei den durchgeführten Usability Tests festgestellt, dass die ersten GUI Prototypen zu wenig ausgereift waren. Die Entscheidungsgeschichte zum Finden des endgültigen User Interfaces ist im Kapitel 7.1 Usability Test beschrieben.

2.3 Zustandsdiagramm

Das nachfolgende Diagramm soll die möglichen Hauptzustände des HOVISSE Workflow Editors, sowie die jeweiligen Übergänge aufzeigen. Diese sind insofern wichtig, da gewisse Vorgänge nur in bestimmten Situationen ausgeführt werden dürfen.

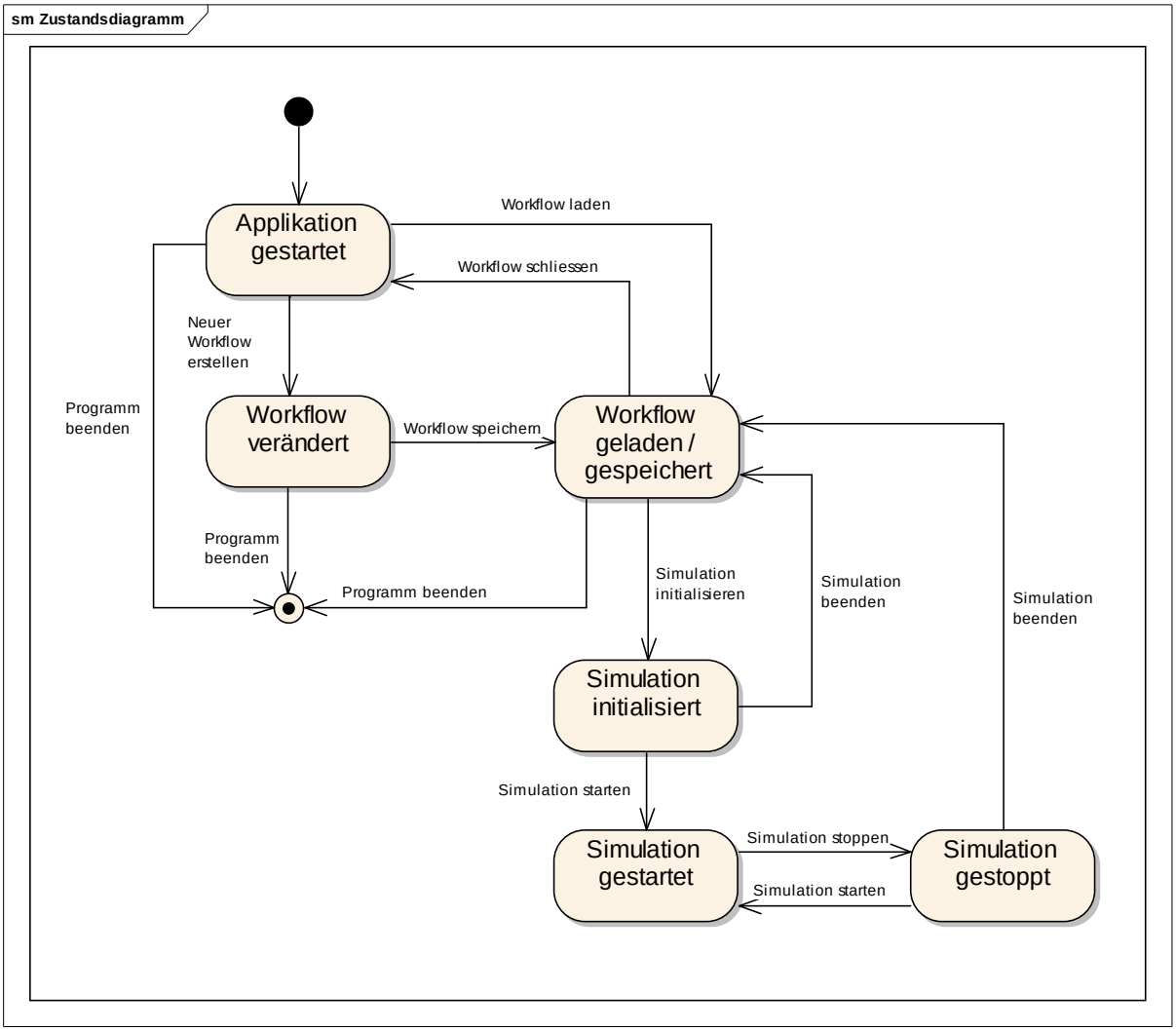

Abbildung 3: Zustandsdiagramm Workflow Editor

2.3.1 Vorhandene Main Menu Punkte pro Zustand

Nachfolgend ist dargestellt, welche Menupunkte zu welchem Zeitpunkt angewählt werden dürfen. Es fällt speziell auf, dass die „Simulation initialisiert“ und „Simulation gestoppt“ die gleichen Aktionen ermöglichen, aber trotzdem unterschieden werden müssen. Dies ist darauf zurückzuführen, dass nach einer abgelaufenen Simulation zusätzliche Schritte ausgeführt werden müssen, bevor eine erneute Simulation gestartet oder in den Edit Modus zurück gewechselt werden kann.

Tabelle 1: Übersicht der möglichen Menupunkte pro Zustand

Main Menu Status	Applikation gestartet (Initial)	Workflow geladen (verändert)	Workflow geladen (unverändert)	Simulation initialisiert	Simulation gestartet (läuft)	Simulation gestoppt
New	X					
Open	X					
Save		X				
SaveAs		X	X			
Close		X	X	X		X
Exit	X	X	X	X		X
Edit Modus				X		X
Sim Modus			X			
Start Workflow				X		X
Stop Workflow					X	
Set Sim Speed				X	X	X

2.3.2 Vorhandene Menu Punkte pro Tree Element Kontext Menu

Nebst dem Main Menu werden je nach Zustand und selektiertem Element im TreePanel entsprechende Kontext Menus angezeigt, um Operationen auf dem Baum auszuführen. Nachfolgend ist dargestellt, welche Kontext Menu Items auf dem jeweiligen Tree Item und Zustand angewählt werden können.

Tabelle 2: Übersicht der angezeigten Kontext Menu Punkte pro Zustand

Tree Context Menu Item	Workflow selektiert (Edit Modus)	Phase selektiert (Edit Modus)	Action selektiert (Edit Modus)	Phase selektiert (Sim Modus)
Add Phase	X			
Add Simple Action		X		
Add Action Group		X		
Add Action Sequence		X		
Delete Phase		X		
Delete Action			X	
Start Phase				X

2.4 Aktivitätsdiagramme

Mit Hilfe der nachfolgenden UML Aktivitätsdiagrammen sollen die Abläufe der wichtigsten, im Pflichtenheft als Use Cases definierten Aktivitäten, dargestellt werden, um so einen besseren Überblick über die möglichen Abläufe zu bekommen.

2.4.1 Programm starten

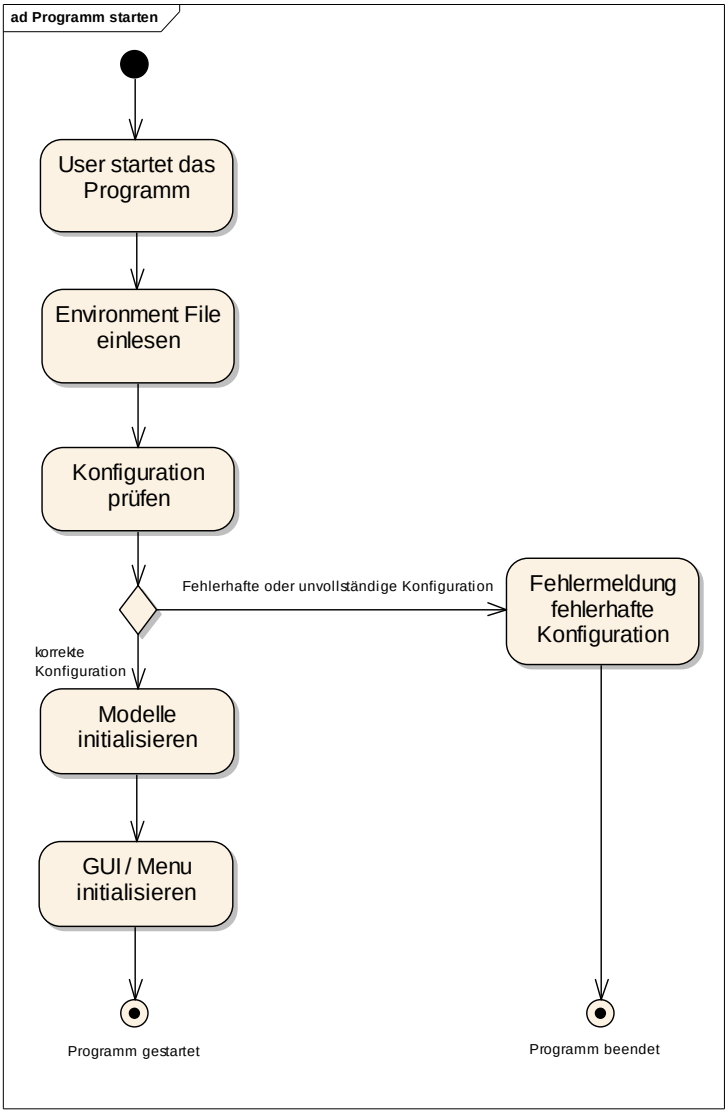

Abbildung 4: Aktivitätsdiagramm Workflow Editor starten

Die allererste Handlung ist dabei das Starten des Editors. Als Nächstes wird das Environment File eingelesen, und falls vorhanden, auf die vollständige und korrekte Konfiguration geprüft. Bei einer unvollständigen oder fehlerhaften Konfiguration wird der Editor sogleich wieder beendet.
Anschliessend erfolgt der Aufbau der Basis Modelle und des Initialen GUI sowie das Erstellen der Menubar. Somit ist der Editor gestartet und zum Arbeiten bereit.

2.4.2 Neuer Workflow erstellen

Eine der beiden möglichen Aktivitäten nach dem Starten des Editors ist das Erstellen eines neuen Workflows. Dabei muss der User ein entsprechendes Inventory definieren. Anhand dessen können dann das Inventory und Catalog File eingelesen werden. Sofern dabei keine Fehler aufgetreten sind, werden die benötigten Workflow und Tree Modelle aufgebaut und schliesslich der Tree sowie das Workflow Panel zur Bearbeitung des Workflows angezeigt.

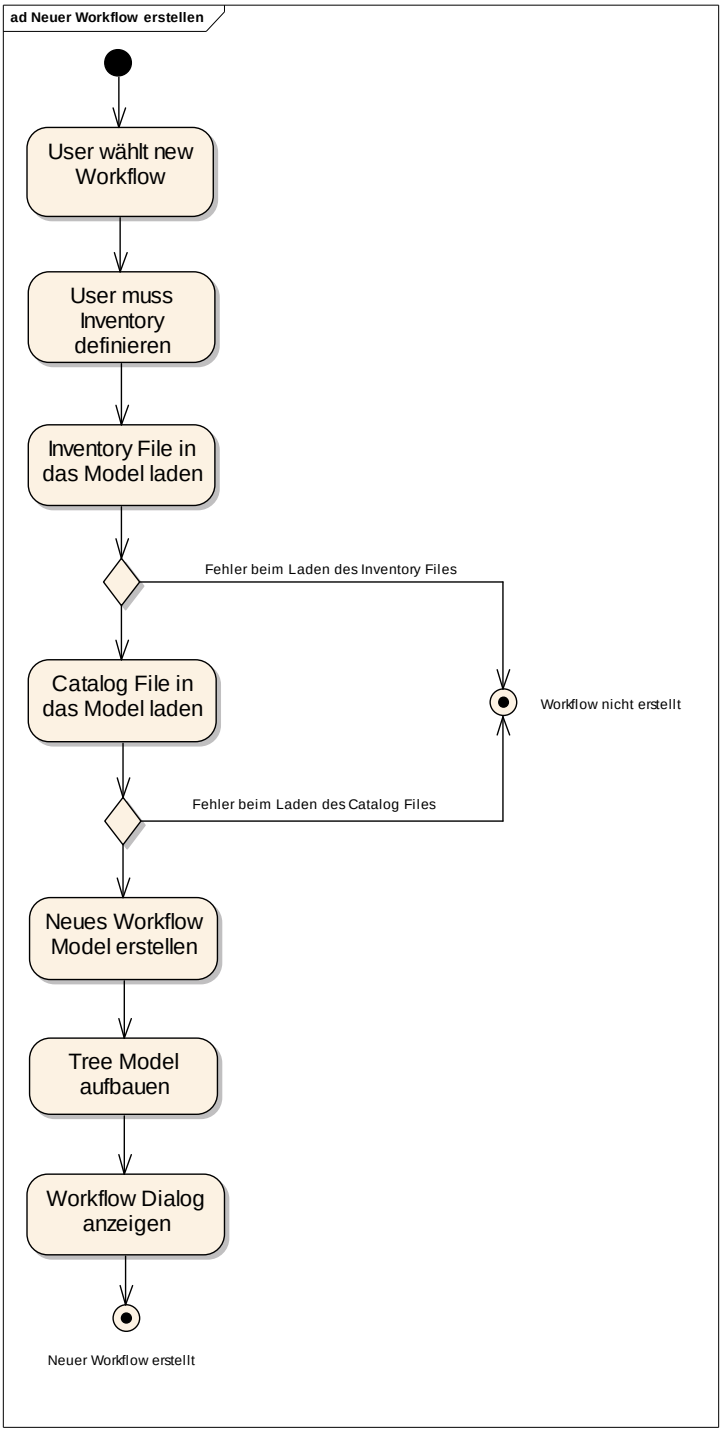
ad Neuer Workflow erstellen
User wählt new Workflow
User muss Inventory definieren
Inventory File in das Model laden
Fehler beim Laden des Inventory Files
Catalog File in das Model laden
Workflow nicht erstellt
Fehler beim Laden des Catalog Files
Neues Workflow Model erstellen
Tree Model aufbauen
Workflow Dialog anzeigen
Neuer Workflow erstellt

Abbildung 5: Aktivitätsdiagramm neuer Workflow erstellen

2.4.3 Workflow laden

Die zweite mögliche Aktivität nach dem Programmstart ist das Laden (Öffnen) eines bestehenden Workflows. Nach dem erfolgreichen Laden des ausgewählten Workflow Files, wird versucht die referenzierten Inventory und Catalog File zu laden. Bei einem Fehler wird der entsprechende Vorgang mit einer Fehlermeldung abgebrochen. Ansonsten wird das Tree Model aufgebaut und der Tree sowie der Workflow Dialog dargestellt.

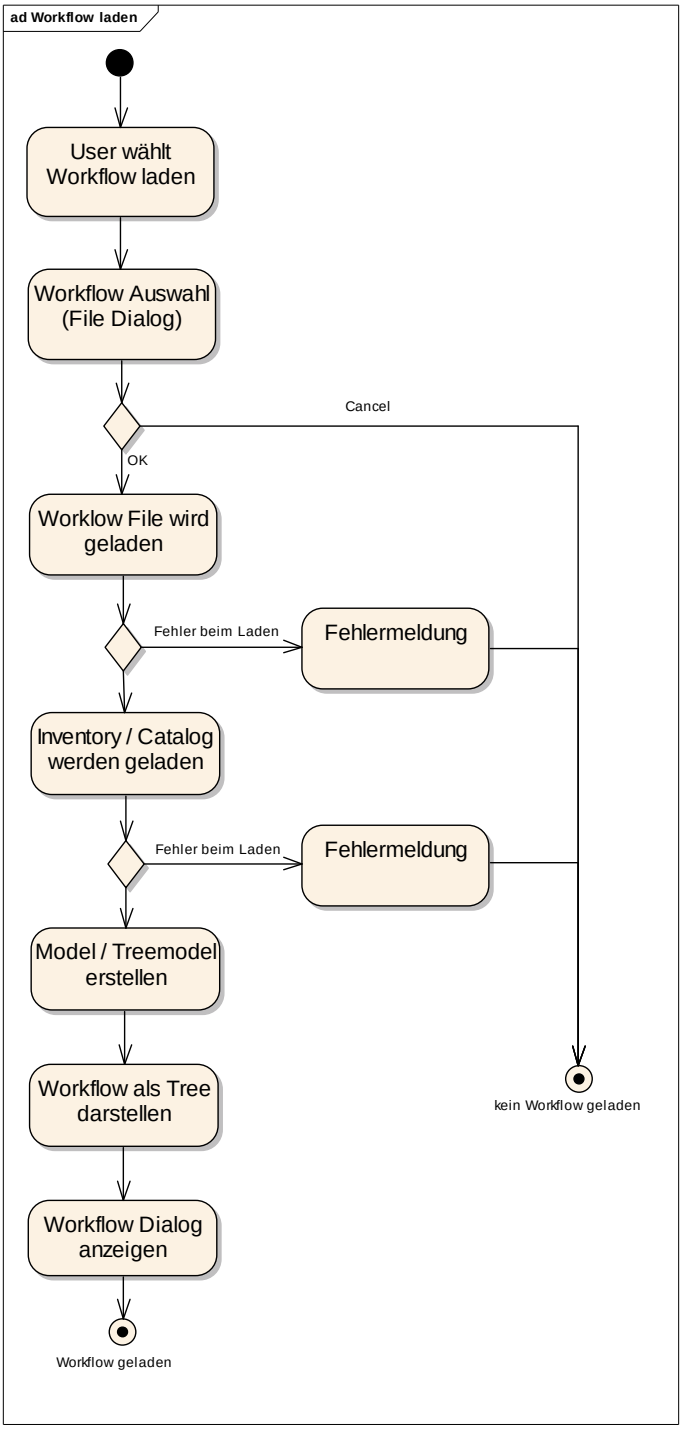

Abbildung 6: Aktivitätsdiagramm Workflow laden

2.4.4 Workflow schliessen

Designbedingt ist es nur möglich einen Workflow gleichzeitig zu bearbeiten. Somit muss der aktuelle Workflow geschlossen werden, bevor ein Neuer erstellt oder geöffnet werden kann.

Beim Schliessen wird überprüft ob der momentan geöffnete Workflow gespeichert wurde, ansonsten wird eine entsprechende Warnung ausgegeben. Andernfalls wird das Model geleert und das GUI bzw. die Menu Punkte wieder entsprechend dem Initialzustand dargestellt.

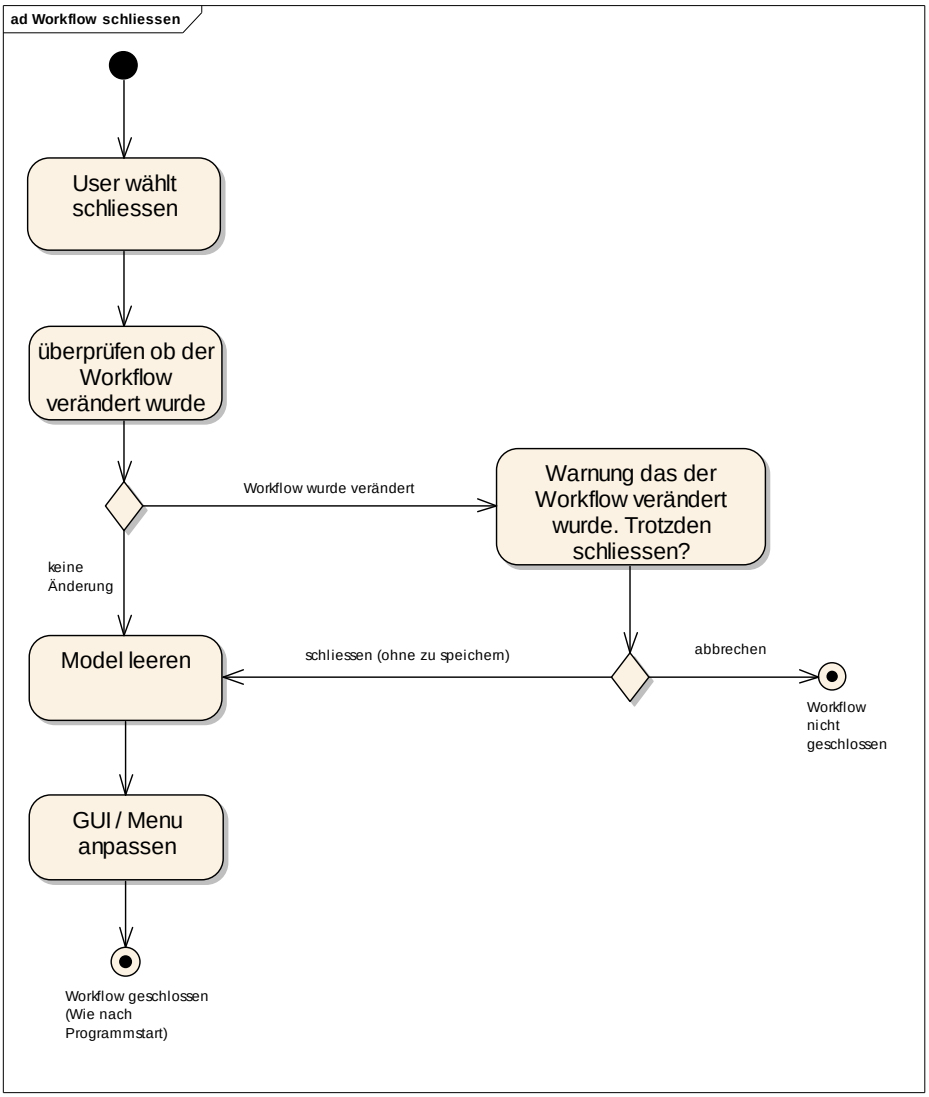

Abbildung 7: Aktivitätsdiagramm Workflow schliessen

2.4.5 Workflow speichern

Einer der wichtigsten Vorgänge ist das Speichern eines erstellten oder veränderten Workflows. Als Erstes wird die Zieldatei, in welche der Workflow gespeichert werden soll ausgewählt. Als vorbeugende Aktivität wird das bestehende File als .old gesichert. Falls das Schreiben (serialisieren) erfolgreich war, werden die Menupunkte entsprechend angepasst, ansonsten wird der Vorgang ohne das File zu Speichern mit einer Fehlermeldung abgebrochen.

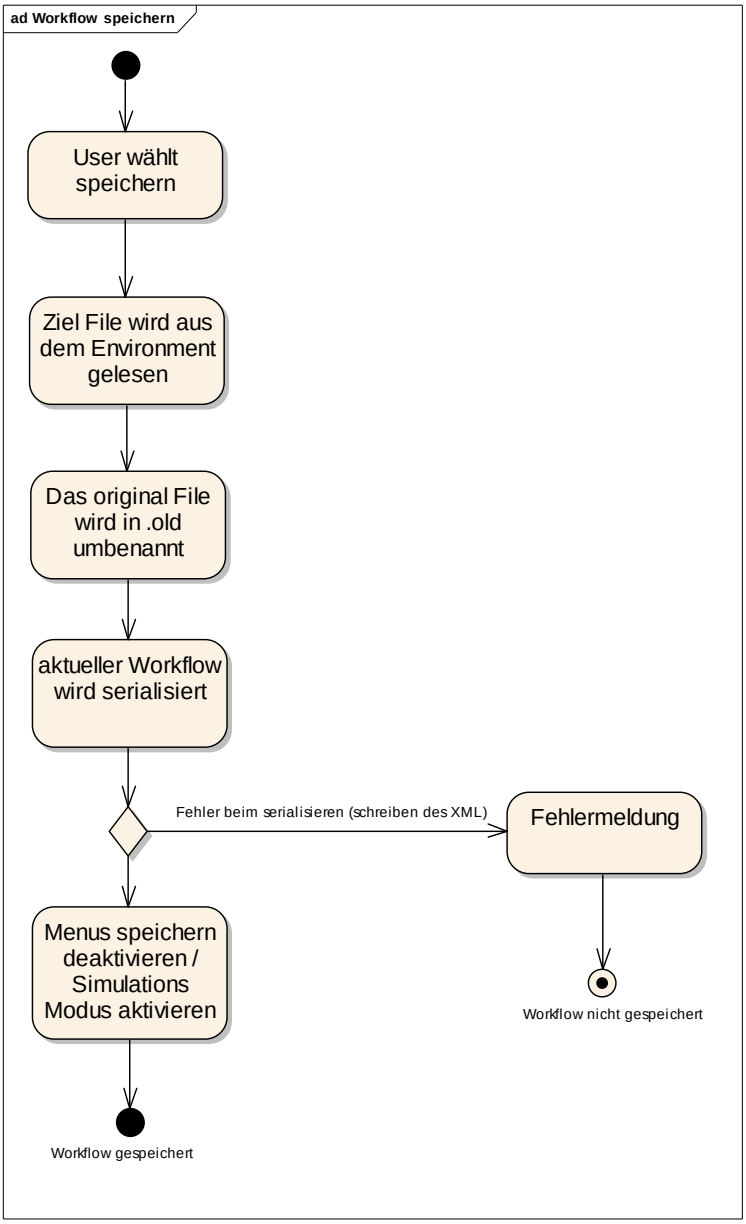
ad Workflow speichern
User wählt speichern
Ziel File wird aus dem Environment gelesen
Das original File wird in .old umbenannt
aktueller Workflow wird serialisiert
Fehler beim serialisieren (schreiben des XML)
Fehlermeldung
Menus speichern deaktivieren / Simulations Modus aktivieren
Workflow nicht gespeichert
Workflow gespeichert

Abbildung 8: Aktivitätsdiagramm Workflow speichern

2.4.6 Simulation Modus anwählen und Simulation starten

Wenn ein Workflow gespeichert ist, kann der Simulation Modus angewählt werden. Dabei wird als Erstes der Simulator initialisiert. Durch das Starten wird die Simulation mit dem aktuellen Workflow oder der gewählten Phase gestartet. Nun werden laufend die aktuellen Proposition sowie die Zustände der vorhandenen Elemente angezeigt, bis die Simulation fertig durchgelaufen ist. Falls dies nicht möglich ist, wird eine entsprechende Fehlermeldung ausgegeben.

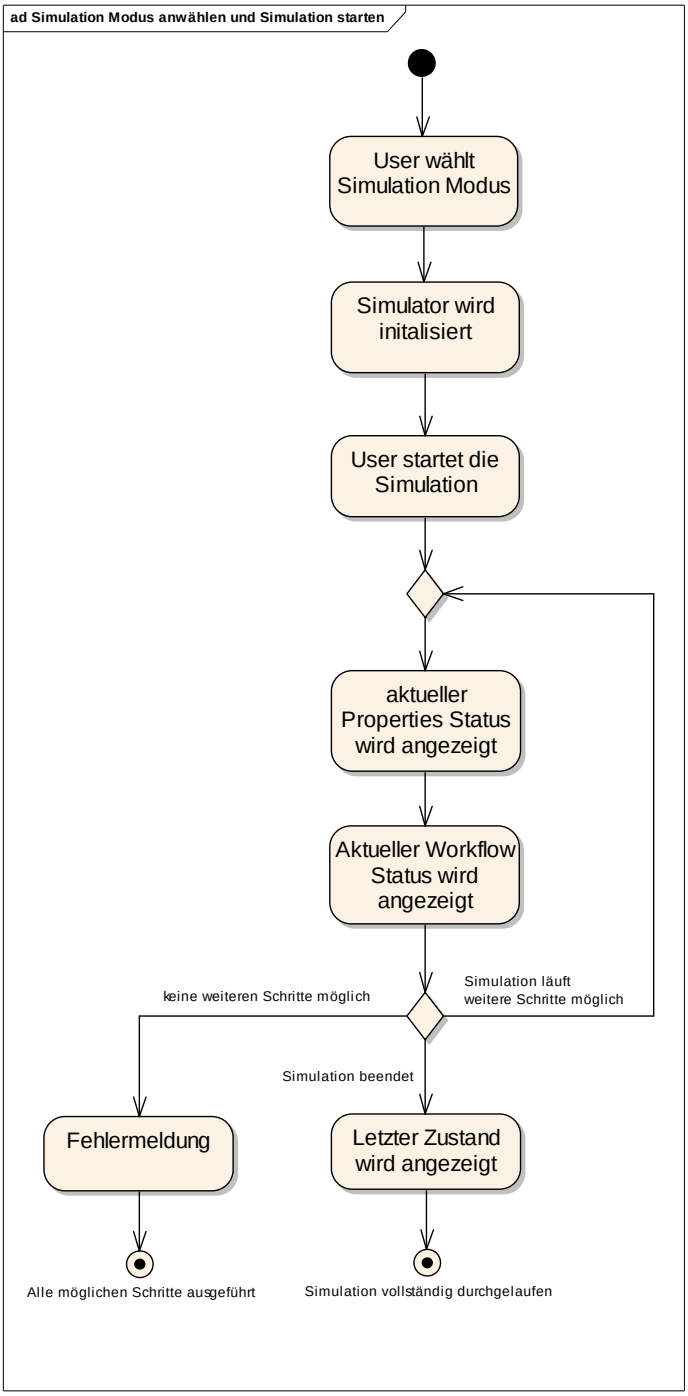

Abbildung 9: Aktivitätsdiagramm Simulation Modus / Workflow simulieren

3 Prototypen

Da im Projekt verschiedene Bibliotheken, mit welchen ich bis jetzt nur wenig oder gar keine Erfahrungen gesammelt hatte eingesetzt werden sollten, habe ich mich dazu entschlossen, die gemachten Überlegungen mit verschiedenen Prototypen zu verifizieren um einerseits die Erfahrungen in das Design einfliessen zu lassen und andererseits allfällige Probleme bereits frühzeitig erkennen zu können.
Bei der Erstellung dieser Prototypen wurde weitgehend auf Architektur und Design verzichtet und alles direkt in eine oder wenige Klassen codiert, um möglichst schnell ein Resultat zu erreichen.

Nachfolgend werden die wichtigsten Prototypen beschrieben, die für dieses Projekt erstellt wurden.

3.1 JAXB

Für das Lesen und Schreiben der XML Files sollte JAXB verwendet werden.
In der nachfolgenden Abbildung sind die Zusammenhänge zwischen Schema, XML File und Klassen/Objekten in JAXB dargestellt.

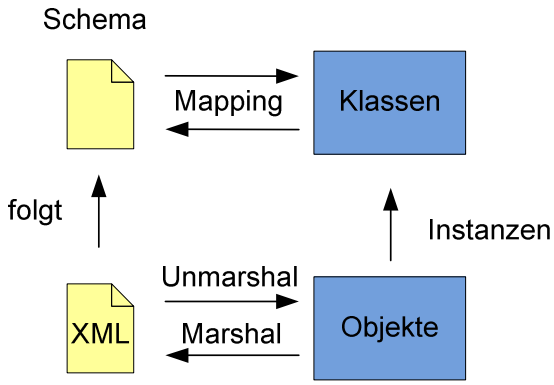

Abbildung 10: Zusammenhänge in JAXB

Aus dem Schema welches das XML beschreibt, können entsprechende Klassen erzeugt werden. Die Unmarshal und Marshal Methoden werden von JAXB zur Verfügung gestellt und ermöglichen das Lesen bzw. Schreiben des entsprechenden XML Files.

Der Prototyp zum Thema JAXB sollte folgenden Fragen beantworten bzw. Schritte testen:

Tabelle 3: Übersicht der Prototype Tests mit JAXB

Testfall	Beschreibung
Klassen generieren	Erzeugen der entsprechenden Java Klassen aus dem vorhandenen Schema mit Hilfe des xjc Kommandos.
XML File einlesen	Mit Hilfe des vorhandenen Unmarshaller ein bestehendes XML einlesen.

Testfall	Beschreibung
Zugriff auf die Objekte	Ausgabe der Objektwerte in das Command Fenster.
Verändern der Objekte	Verändern von einfachen und komplexen Werten, sowie Elementen innerhalb von Listen bzw. Elemente aus der Liste entfernen und neue hinzufügen.
Schreiben des XML Files	Mit Hilfe des vorhandenen Marshaller ein XML File zurückschreiben und überprüfen, ob die veränderten Werte richtig zurückgeschrieben wurden.

Diese Tests konnten alle ohne Probleme durchgeführt werden. In der verwendeten Java Version 6.0 ist JAXB 2.0 bereits enthalten, so dass keine zusätzliche Library benötigt wurde.

Dabei sind folgende Beobachtungen gemacht worden:

- Wichtig ist es den korrekten Package Path für die zu erzeugenden Klassen einzusetzen.
- Ein Problem wurde im Zusammenhang mit dem dtd File festgestellt. Es muss sich im gleichen Pfad wie das XML File befinden. Ich habe keinen Weg gefunden, einen alternativen Pfad für das dtd File zu definieren.

3.2 Menu Handling

Mit dem Menu Prototype wurde die Verwendung der AbstractAction Klasse für die Erstellung eines Menus getestet. Dabei wird das Menu mit Hilfe von Action Instanzen aufgebaut. Als Reaktion auf einen ausgewählten Menupunkt muss einzig die actionPerformed Methode der jeweiligen Action implementiert werden.

Dies wurde entsprechend umgesetzt und in die actionPerfomed Methode jeweils eine Ausgabe auf den Cmd Prompt geschrieben. Zusätzlich wurde das enablen und disablen von Menupunkten getestet. Dies ist nachfolgend dargestellt und hat wie vorgesehen funktioniert.

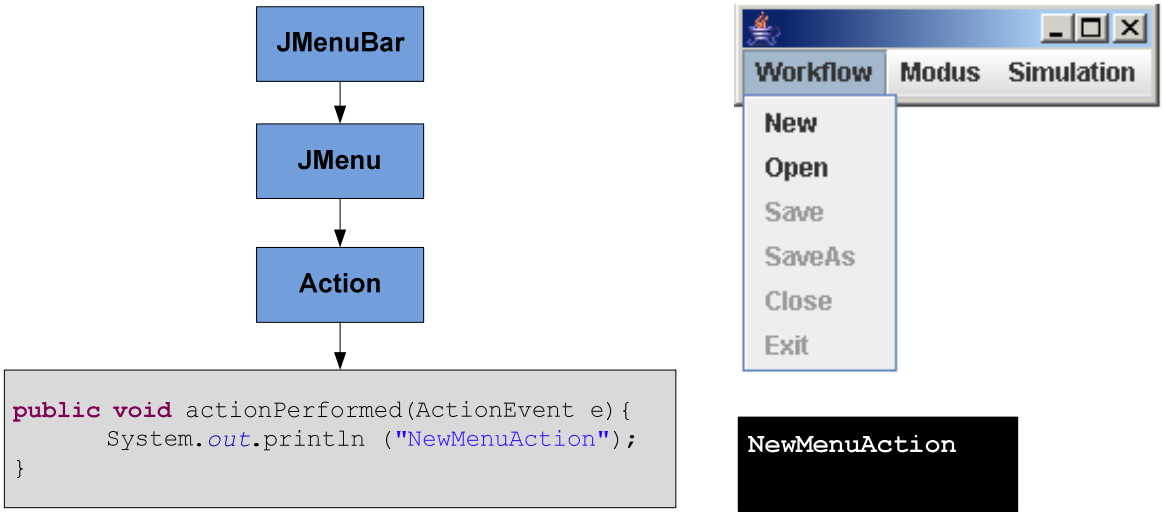

Abbildung 11:Aufbau eines Menus mit Actions & Beispiel Menu mit Sample Output

3.3 JTree

Tabelle 4: Übersicht der Prototype Tests für JTree

Testfall	Beschreibung
Anzeigen eines Trees	Aufbauen und Darstellen eines JTrees mit Swing.
Selektieren eines Elements	Anwählen eines Elements im Tree. Es wird ein Event ausgelöst und das Element erhält den Fokus.
Löschen und Hinzufügen von Elementen	Im Tree wird das neue Element angezeigt bzw. das gelöschte Element nicht mehr.
Kontext Menu hinzufügen	Auf den unterschiedlichen Elementen wird bei einem Rechtsklick das entsprechende Kontext Menu anzeigt.

Diese Tests konnten grösstenteils problemlos durchgeführt werden. Die nachfolgenden Beobachtungen sind dabei gemacht worden:

- Standardmässig können mehrere Elemente selektiert werden. Dies war in meinem Fall jedoch unerwünscht und musste speziell eingeschränkt werden.
- Auf der rechten Maustaste ist das Kontextmenu hinterlegt. Wenn jedoch mit der rechten Taste ein Element angewählt wird, erhält das selektierte Element keinen Fokus, dieser muss vorgängig selber gesetzt werden.
- Beim Löschen eines Elements geht der Fokus verloren. Da aber ein aktuelles Element angewählt sein muss, setzte ich den Fokus in diesem Fall auf das Parent Element.
- Beim Einfügen eines neuen Elements bleibt der Fokus verständlicherweise auf dem Element auf welchem die Operation ausgeführt wurde. Der Fokus muss entsprechend auf das neu hinzugefügte Element gesetzt werden.

3.4 Panel Prototype

Um nicht alle Dialoge / Panels von Hand erstellen zu müssen, wollte ich nach Möglichkeit den Mantisse GUI Builder aus Netbeans verwenden.
Somit ging es bei diesem Prototype einerseits darum die generierten Panels nach Eclipse zu übernehmen und anzuzeigen. Andererseits soll je nach angewähltem Element im Tree das entsprechende Panel im Scroll Panel angezeigt werden. Dabei ging es auch darum zu überprüfen, ob der Austausch der Panels schnell genug erflogen kann.

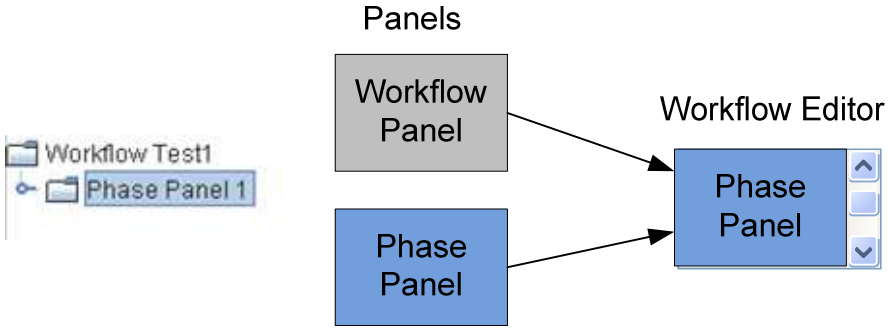

Abbildung 12: Austauschen eines Panels

Die Übernahme der Panels ist grundsätzlich problemlos möglich. Der Mantisse Builder in NetBeans scheint aber noch einige Mängel aufzuweisen:

- Für die Beschriftung von Buttons habe ich die Sonderzeichen für Pfeile verwendet. In der Preview Darstellung waren diese entsprechend korrekt. Nach der Übernahme nach Eclipse musste jedoch die entsprechende Schriftart und die entsprechenden Sonderzeichen von Hand korrigiert werden.
- Je nach Hinzufügen oder Verschieben von Elementen im Editor, ändert sich die Grösse des Fensters oder das Panel hat nicht mehr eine fixe Grösse sondern ist fliessend. Dies lässt sich nicht mehr rückgängig machen. Somit sollte vor „heiklen“ Veränderungen zwischengespeichert werden.

3.5 Test der Workflow Engine

Beim Test der Workflow Engine wollte ich die Anbindung an die erhaltene Workflow Engine testen.

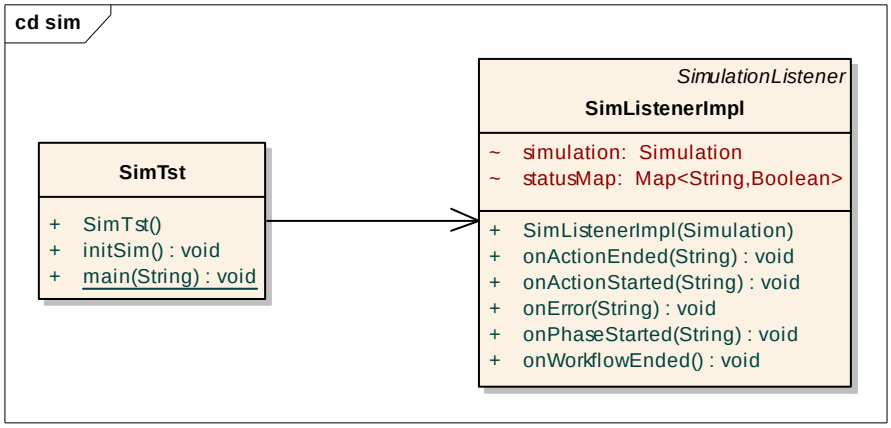

Abbildung 13: Testprogramm für die Workflow Engine

Die Implementierung des SimListener bestand wiederum darin, für die jeweils erhaltenen Events entsprechende Meldungen auf den Command Prompt zu schreiben.

```
PhaseStarted phaseId p1
ActionStarted actionId T1
ActionStarted actionId T2
ActionEnded actionId T1
----Property STATUS ---
P1=true
P2=true
P7=false
```

Abbildung 14: Test Output

Dieser Test konnte ebenfalls erfolgreich durchgeführt werden.

4 Design

4.1 Einführung

Die benötigten Klassen bzw. Pakete werden in die nachfolgenden vier Layer eingeteilt. Dabei sollen die Layer von unten nach oben aufbauend sein.

- Der Support Layer enthält unterstützende Klassen.
- Der Data Layer enthält die Klassen für die eigentlichen Daten Objekte sowie für den Zugriff (lesen und schreiben) der Daten (XML Files).
- Der Business Layer enthält die Logik Klassen. Da es sich um ein GUI Projekt handelt ist dieser Anteil hier recht gering.
- Der Presentation Layer enthält sämtliche Klassen im Zusammenhang mit dem GUI. Dies sind die Panels welche für die Darstellung und Bearbeitung der unterschiedlichen Daten verwendet werden, sowie zusätzlich benötigte Dialoge.

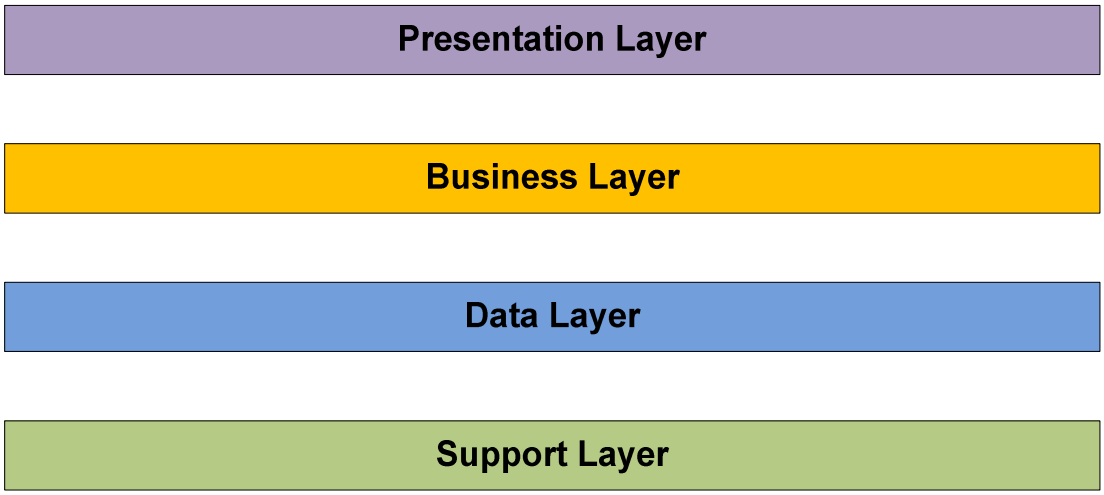

Abbildung 15: Software Layer

4.2 Klassendiagramm (Übersicht)

Als Erstes soll mit dem nachfolgenden, groben Klassendiagramm eine erste Übersicht über die vorhandenen Pakete und Klassen vermittelt werden.

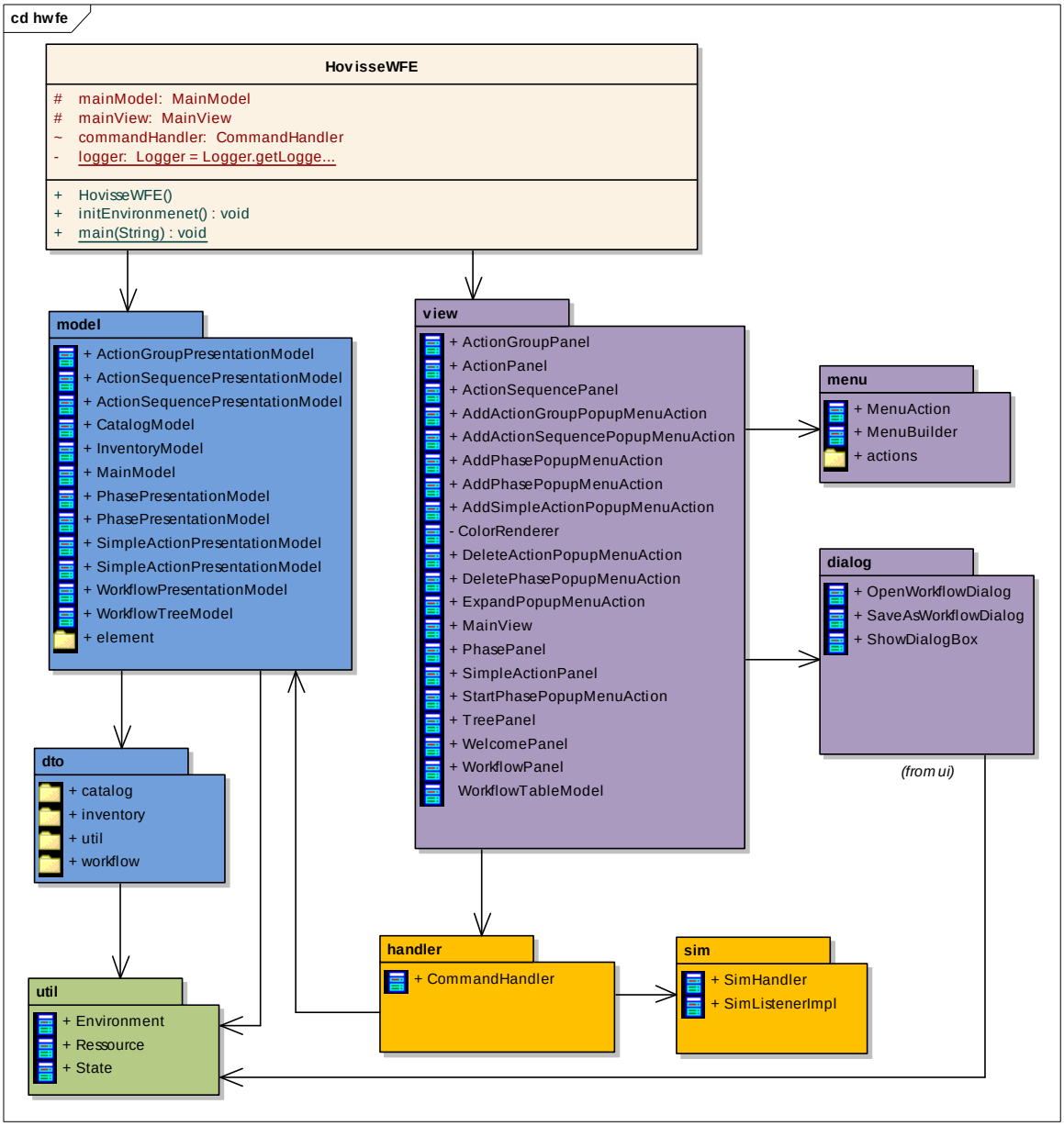

Abbildung 16: Software Layer

Die beiden Pakete auf welchen der HOVISSE Workflow Editor aufbaut, beinhalten die Model und die View Klassen. Die View benutzt die zum Presentation Layer gehörenden Pakete Menu und Dialog. Das Model benutzt die dto Klassen, welche im Prinzip das Datenmodell darstellen. Der Handler setzt die durch die View ausgelöste Handlung im Modell um. Die Simulation reagiert auf die Informationen, welche sie von der Workflow Engine erhält entsprechend. Nachfolgend sind nun die einzelnen Pakete und Klassen genauer beschrieben.

4.3 Support Layer

4.3.1 Pakete

Der Support Layer enthält unterstützende Pakete welche von allen Klassen verwendet werden können.

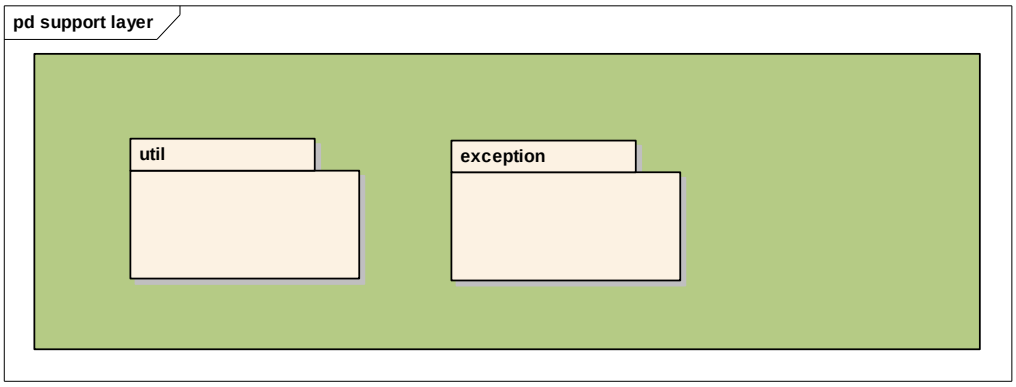

Abbildung 17: Software Layer Pakete

Tabelle 5: Übersicht Support Layer Pakete

Paket	Beschreibung
util	Beinhaltet die Environment Klasse, welche die Umgebung verwaltet, sowie eine State Enumeration, welche die Zustände verwaltet.
exception	Beinhaltet die eigenen Exception Klassen.

4.3.2 Environment Klasse

Die Environment Klasse verwaltet, wie der Name schon zu erkennen gibt, die Umgebung. Sie liest das Properties File und prüft es auf dessen Vollständigkeit. Die Environment Klasse kennt dadurch die Pfade und Filenamen zu den aktuell geöffneten und benutzten Workflow, Inventory und Catalog Files, bzw. zum default Workflow Verzeichnis.
Des Weiteren enthält die Environment Klasse eine Enumeration für die Zustände der Applikation, also ob ein Workflow geladen oder gespeichert ist oder momentan eine Simulation am Laufen ist. Diese wird zum Steuern der Menuitem und Kontextmenus verwendet. Die Environment Klasse ist statisch, so dass jederzeit auf sie zugegriffen werden kann.

4.3.3 Exception Klassen

Im Package Exception befinden sich zudem noch zwei spezifische Exception Klassen für Fehler, bei welchen nicht auf eine vorhandene Exception zurückgegriffen werden konnte.

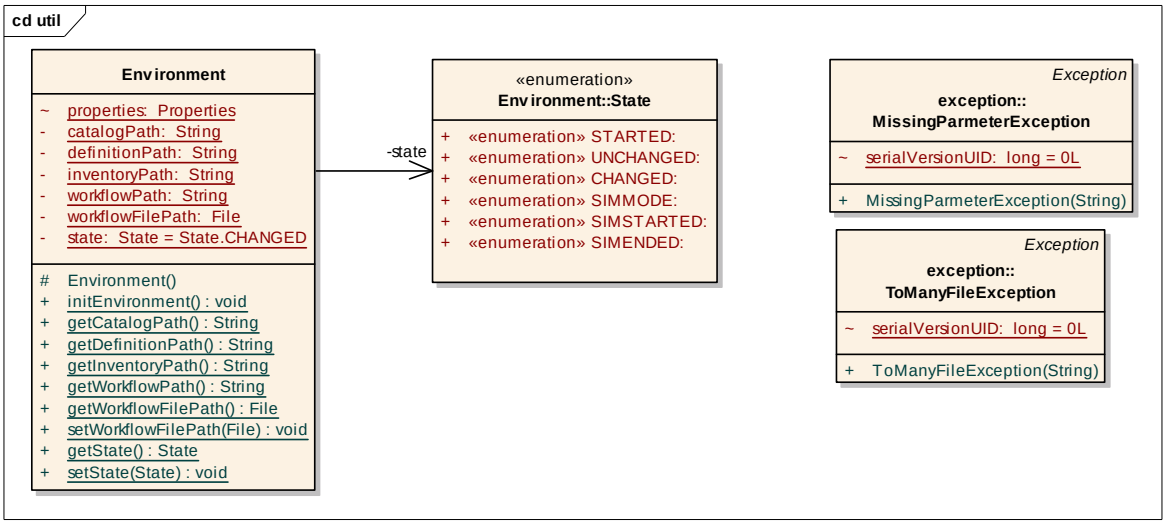

Abbildung 18: Environment und Exception Klassen

4.4 Data Layer

4.4.1 Pakete

Die Pakete des Data Layer können im Prinzip wiederum in drei Ebenen aufgeteilt werden. Das dto.util Paket ist für den Zugriff und den Austausch der Daten mit dem XML File verantwortlich. Die dto.workflow, dto.inventory und dto.catalog Pakete stellen die eigentlichen Daten dar. In den Model Klassen werden die bestehenden Daten zur vereinfachten Verarbeitung oder Darstellung optimiert.

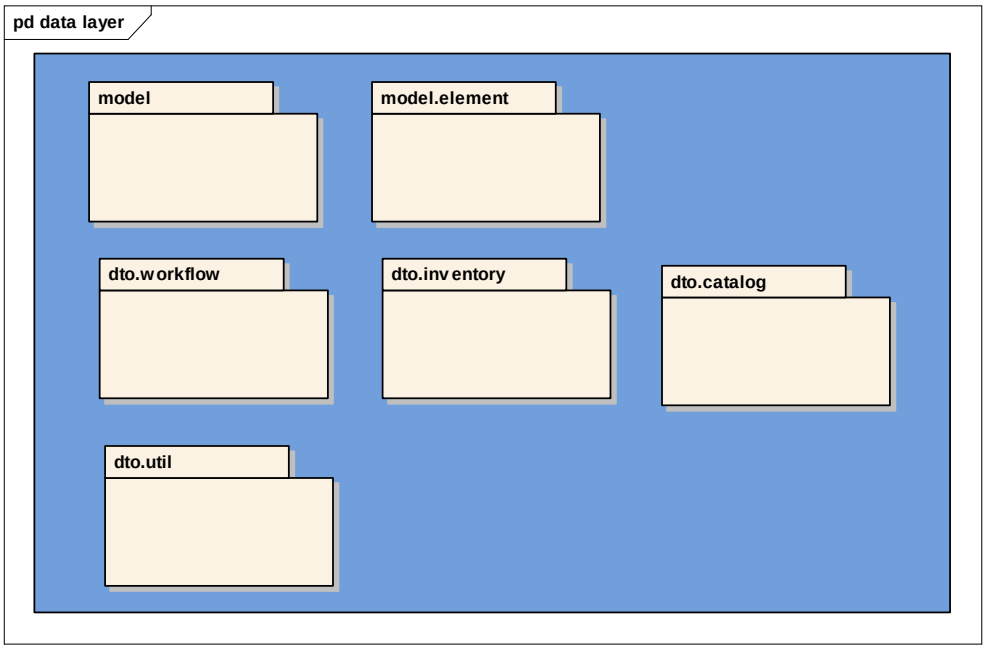

Abbildung 19: Data Layer Pakete

Tabelle 6: Übersicht Data Layer Pakete

Paket	Beschreibung
dto.workflow	Beinhaltet die mit JAXB generierten Workflow Klassen.
dto.inventory	Beinhaltet die mit JAXB generierten Inventory Klassen.
dto.catalog	Beinhaltet die mit JAXB generierten Catalog Klassen.
dto.util	Beinhaltet die Klassen zum Einlesen und Speichern der dto Klassen.
Model	Beinhaltet die verschiedenen Daten Modelle und die entsprechenden Methoden um die Modelle anzupassen.

4.4.2 Data Model

Das Mainmodel kennt und erzeugt alle anderen Models. Nachfolgend ist der erste Teil dargestellt, welcher auf den dto Klassen basiert. Das Catalog Model stellt bessere Zugriffmethoden auf die Daten zur Verfügung. Somit können mit Hilfe des Keys (id) die verschiedenen Elemente abgerufen werden. Damit bei den Listelementen nicht bei jeder Abfrage die ganze Liste durchsucht werden muss, sollen die Element Referenzen in Hashtables abgefüllt werden.

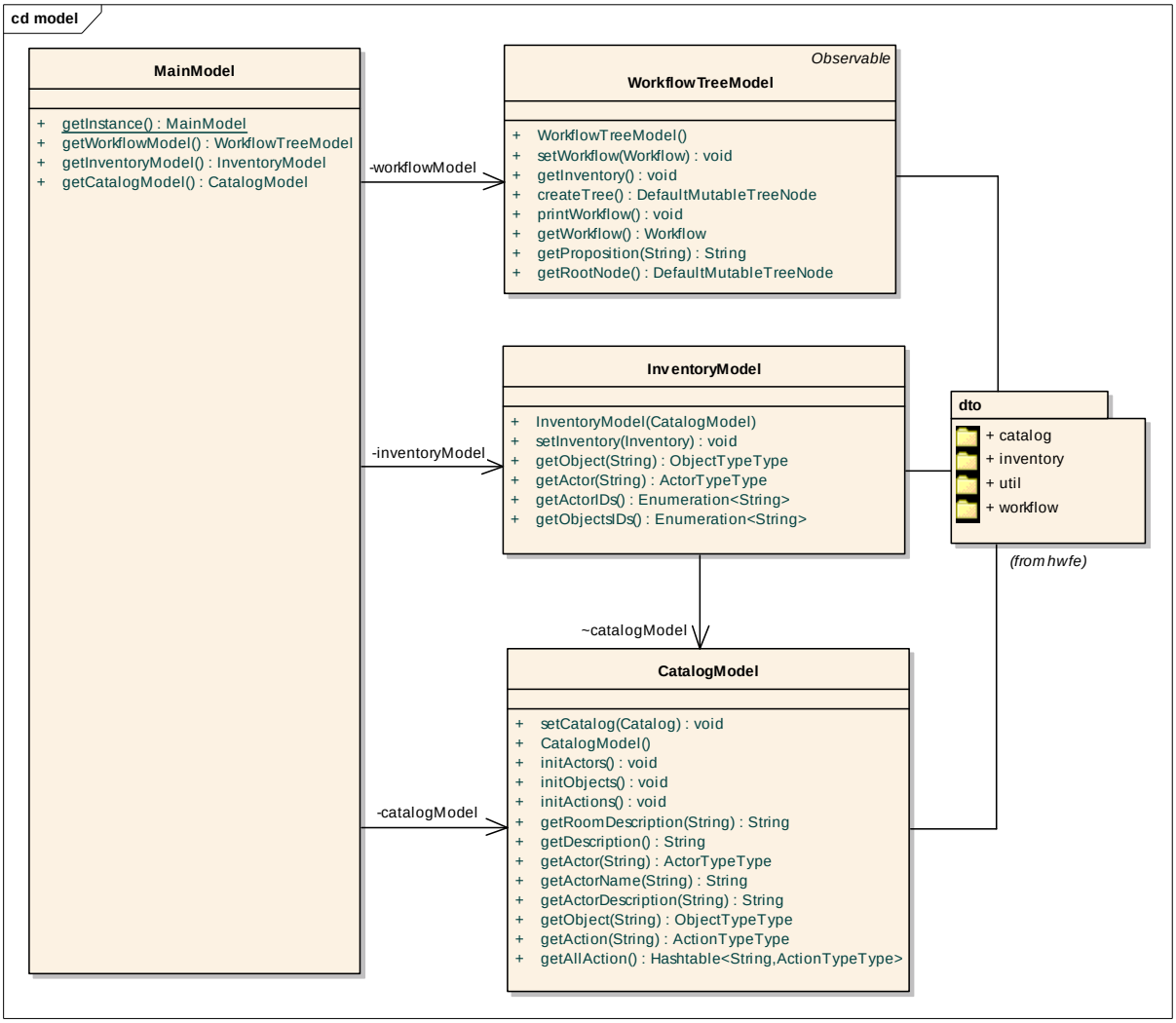

Abbildung 20: Data Model Klassen

Beim Inventory ist es ähnlich, dort sollen jedoch zusätzlich gewisse Referenzen zum Catalog bereits aufgelöst werden. Somit muss bei einer Referenzanfrage nicht auch noch zusätzlich der Catalog abgefragt werden.
Das WorkflowTreeModel schliesslich ist bereits eine spezielle Aufbereitung für die Anzeige im JTree. Mit Hilfe von DefaultMutableTreeNode Objekten wird eine Baumstruktur aufgebaut. Der Inhalt der Objekte ist dabei eine eigene Element Klasse. Der Vorteil dabei ist, dass so jedes Element gleich behandelt werden kann.

4.4.3 Presentation Model

Die Presentation Model Klassen sind sozusagen die Wrapper Klassen zwischen dem GUI und dem Data Model. Sie stellen einerseits die Schnittselle zum GUI dar und bereiten die Daten so auf, dass sie möglichst einfach dargestellt werden können.
Andererseits hält die jeweilige Presentation Model Klasse die Referenzen auf die Datenmodelklassen und ist in der Lage diese entsprechend der Rückmeldung des GUI zu aktualisieren. Die entsprechenden Klassen werden für jedes Objekt bei Bedarf neu erzeugt.

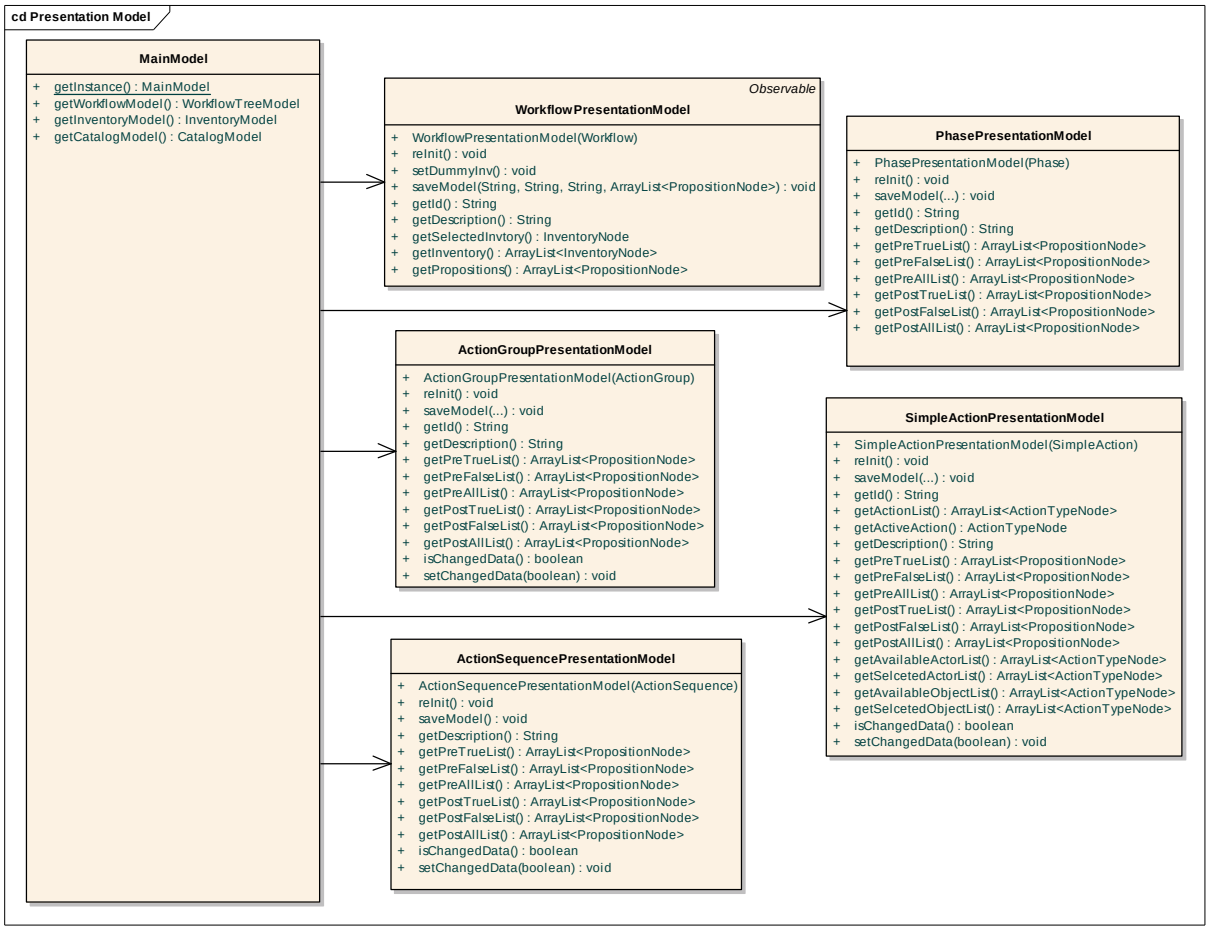

Abbildung 21: Presentation Model Klassen

4.4.4 dto workflow

Die dto workflow Klassen wurden aus dem xsd File mit Hilfe von JAXB generiert. Sie sind die sogenannten Daten Transfer Objekte (dto), da die entsprechenden Instanzen ein Objekt Abbild des XML Files darstellen. Mit Hilfe dieser Klassen werden die entsprechenden Modelle aufgebaut. Dies stellt im Prinzip ein Teil des Datenmodels dar.

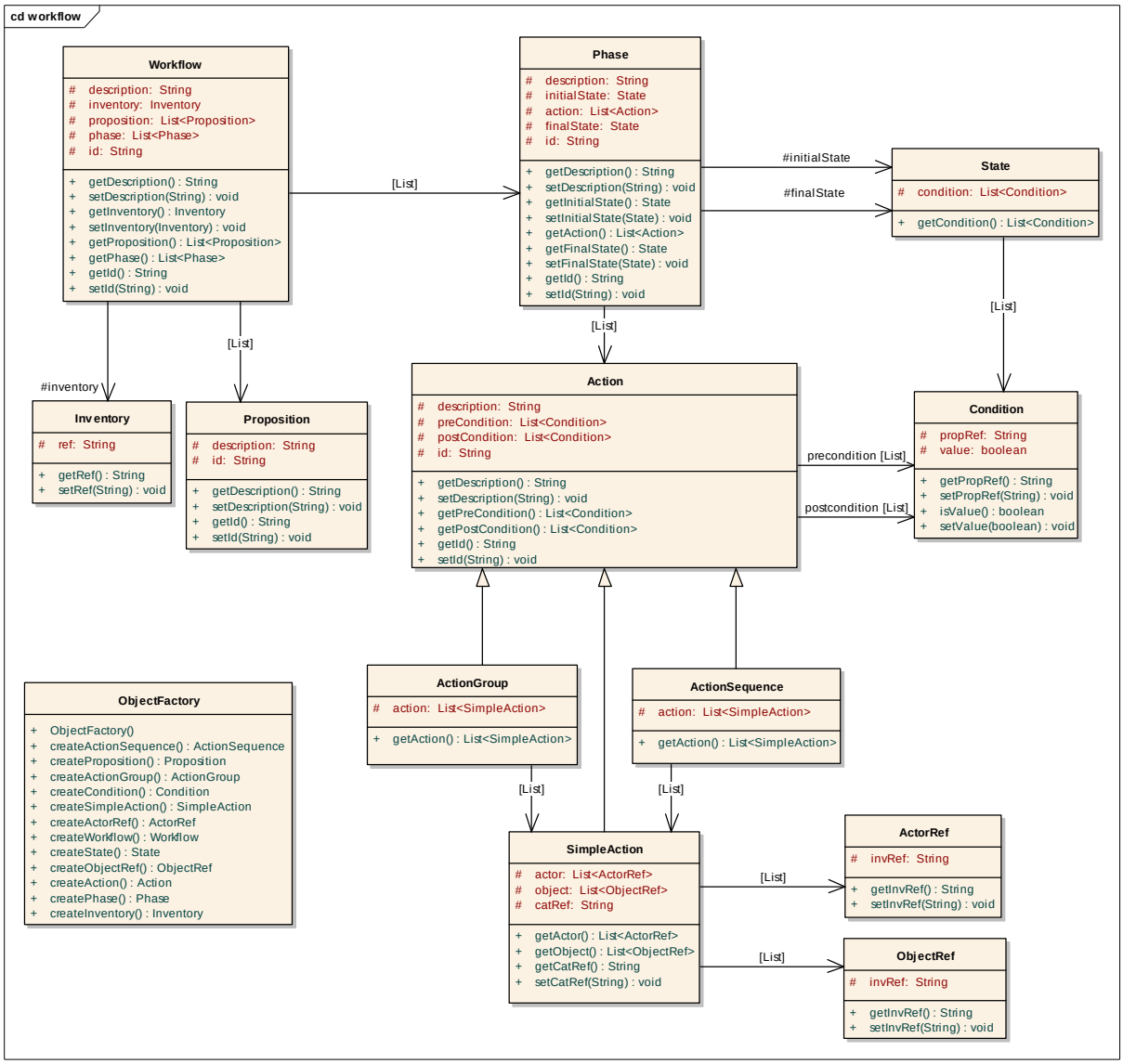

Abbildung 22: dto.workflow Klassen

Die Abhängigkeiten zur ObjectFactory wurden zur besseren Übersicht nicht eingezeichnet, da diese die entsprechenden Objekte erzeugt und somit zu allen anderen Klassen Abhängigkeiten aufweist.
Diverse hier zu sehende Objekte weisen Abhängigkeiten zum Inventory und Catalog auf.

4.4.5 dto inventory

Die dto inventory Klassen wurden ebenfalls generiert. Sie stellen die Menge der verfügbaren Elemente pro Type zur Verfügung. So kann einerseits das vorhandene Subset definiert werden und andererseits die Anzahl der Elemente, welche von einem gewissen Objekt (z.B. mehrere Handschuhe) vorhanden sind. Dies ist ebenfalls ein Teil des Datenmodels.

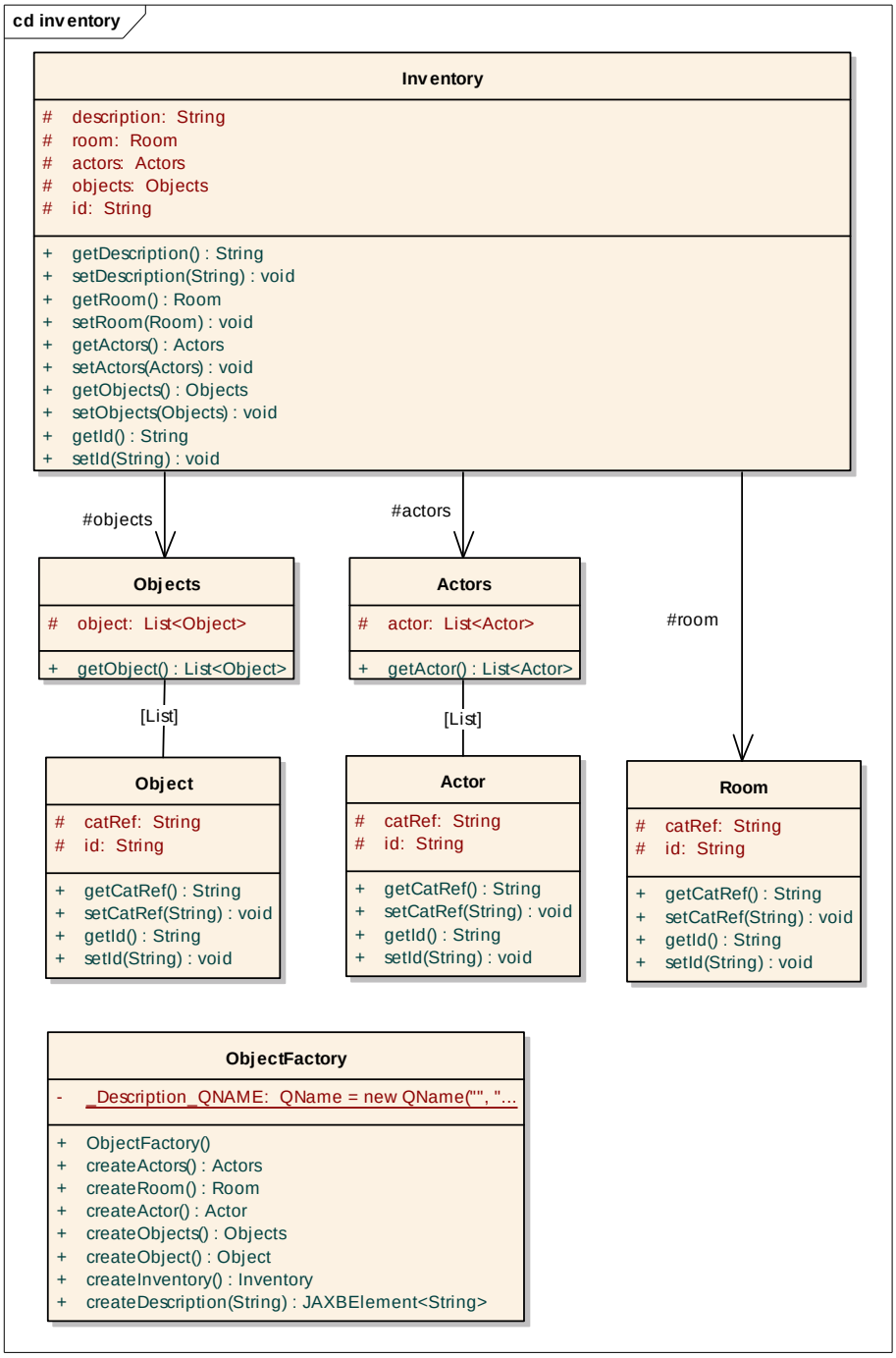

Abbildung 23: dto.inventory Klassen

4.4.6 dto catalog

Die dto catalog Klassen stellen den dritten Teil des „Datenmodels“ dar und wurden wie die Klassen der beiden vorangehenden Pakete ebenfalls generiert. Die dto catalog Klassen stellen die Menge der verfügbaren Typen von Objekten zur Verfügung.

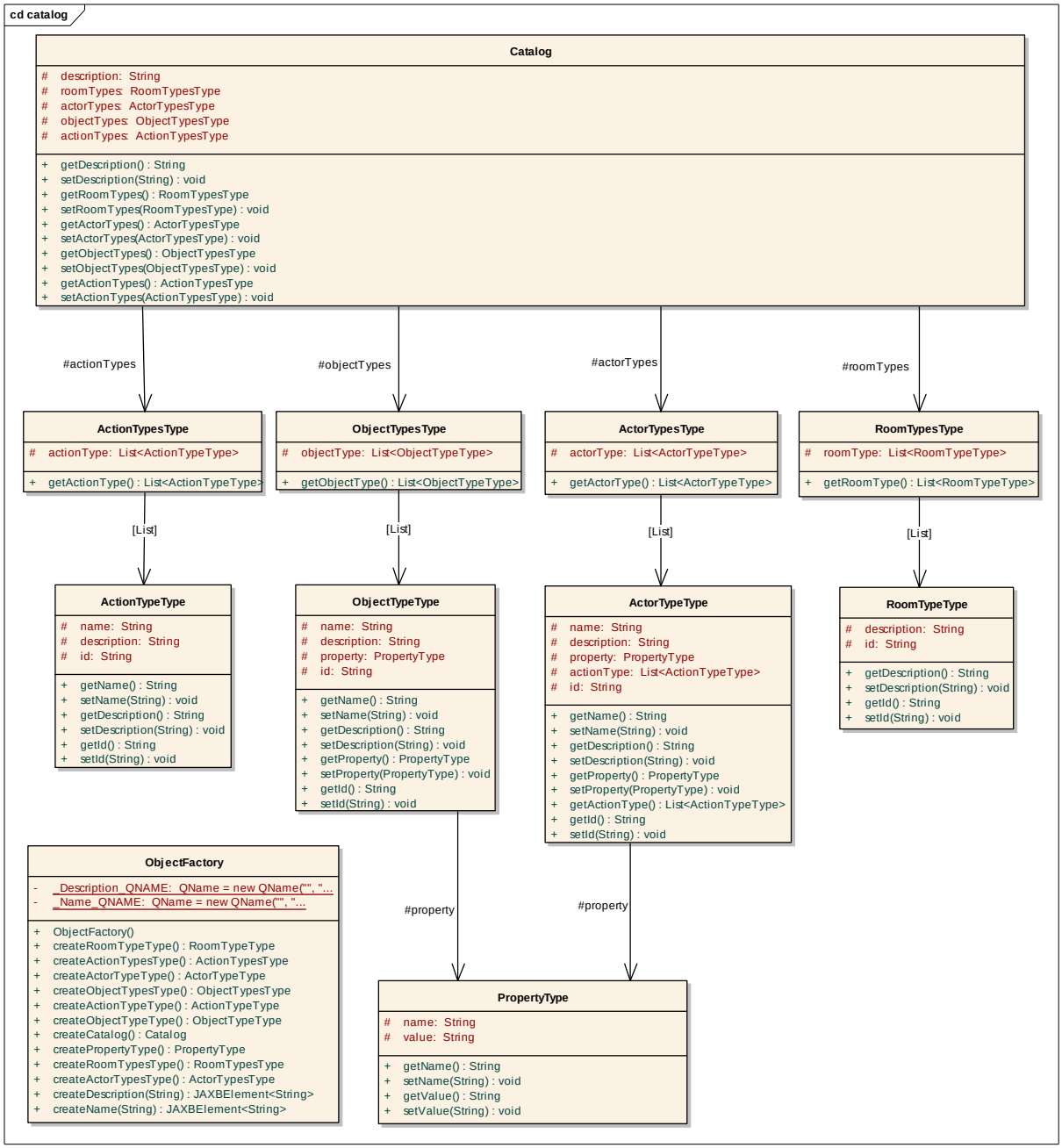

Abbildung 24: dto.catalog Klassen

4.4.7 dto.util

Die drei dto.util Klassen sind die serializer bzw. deserializer Klassen, dass heisst sie lesen die Daten mit Hilfe der JAXB Technologie in das generierte Objekt Model, bzw. schreiben das Objektmodell wieder in die XML Struktur zurück.
Des Weiteren bieten Sie die Möglichkeit, das XML File bzw. die Objektstruktur gegen das entsprechende Schema zu validieren.
Catalog und Inventory müssen im Prinzip nur gelesen werden. Der Vollständigkeit halber stehen die Methoden zum Serializieren der jeweiligen Objektmodelle ebenfalls zur Verfügung.

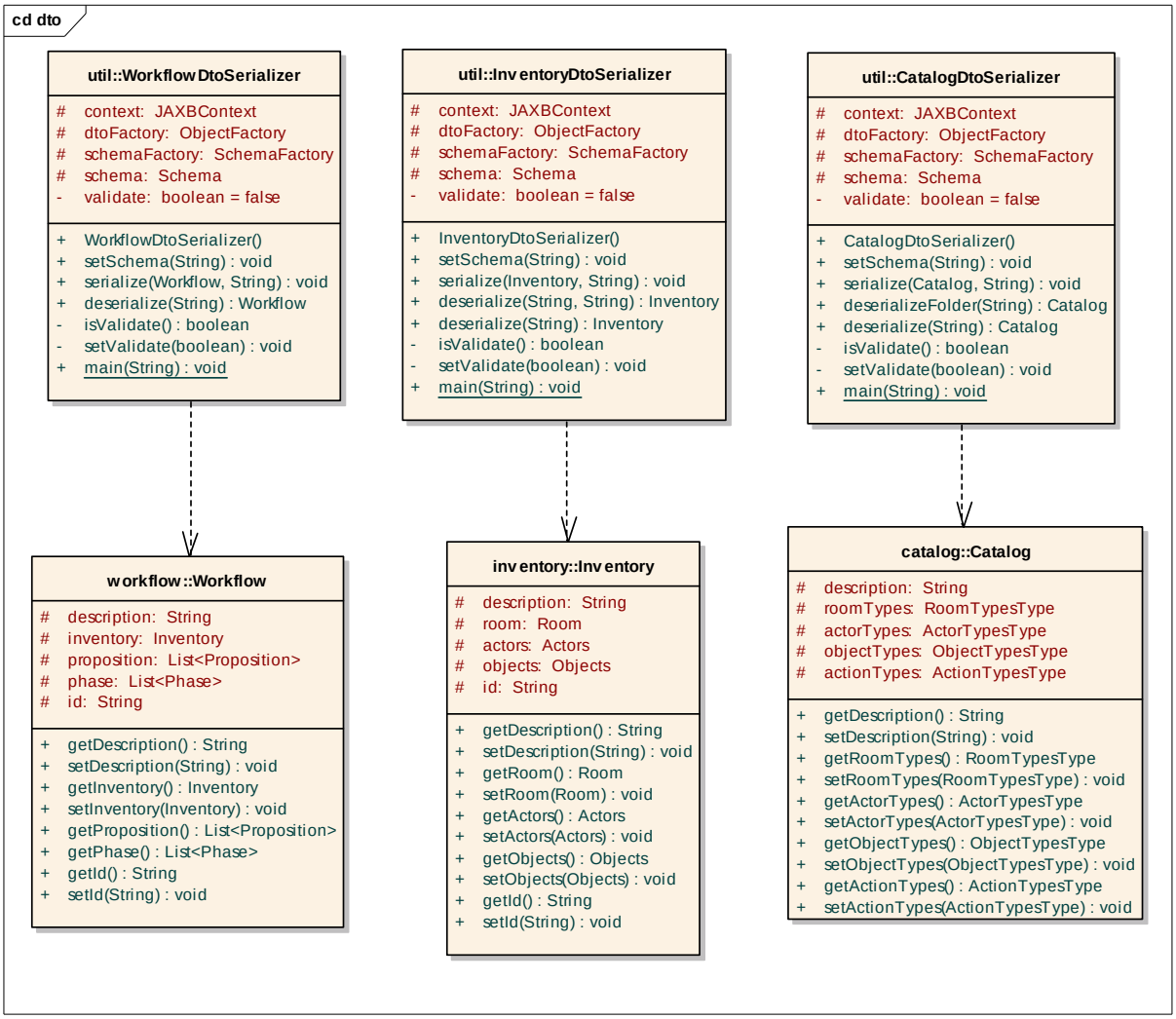

Abbildung 25: dto.util Klassen

4.5 Business Layer

4.5.1 Pakete

Der Business Layer enthält nur die zwei Pakete, welche die eigentliche Applikationslogik enthalten. Der Anteil an Business Logik ist in diesem Projekt relativ gering.

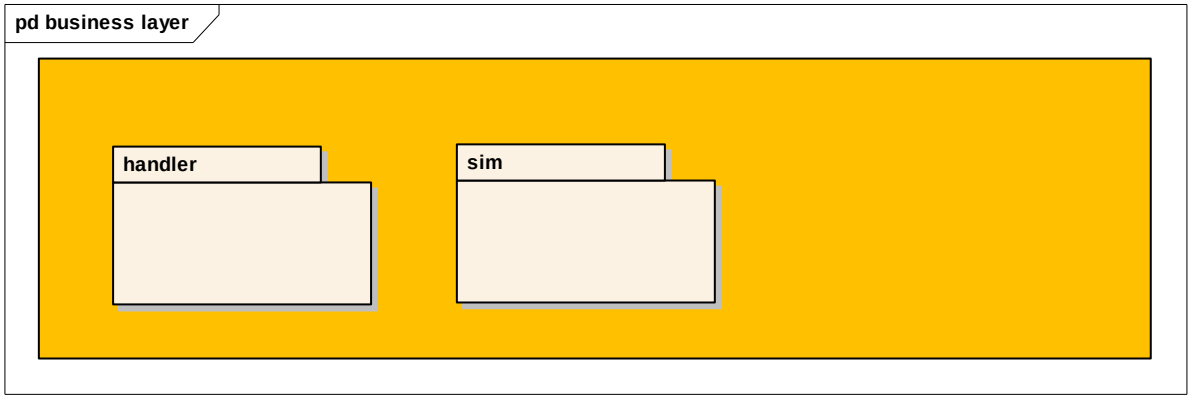

Abbildung 26: Business Layer Pakete

Tabelle 7: Übersicht Presentation Layer Pakete

Paket	Beschreibung
handler	Beinhaltet die Command Handler Klasse.
sim	Beinhaltet die Klassen welche auf die Antworten der Workflowengine reagieren.

4.5.2 Command Handler Klasse

Der Command Handler wird aufgrund einer User Interaktion (z.B. open workflow) aufgerufen. Er führt die notwendigen Schritte durch, um die gewünschte Aktivität auszuführen.

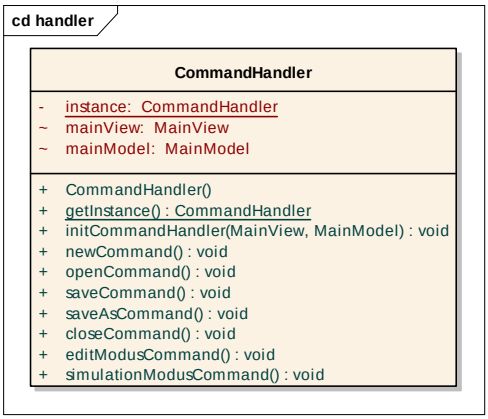

Abbildung 27: Command Handler Klasse

4.5.3 Simulations Klassen

Die SimHandler Klasse ist für die Initialisierung der Simulation zuständig, sowie für das Starten und Stoppen. Des Weiteren kann die gewünschte Simulations-geschwindigkeit festgelegt werden.
Die SimListenerImpl Klasse ist die Listener Klasse, welche auf die Meldungen der Workflow Engine hört. Entsprechend der gemeldeten Zustände werden die Informationen in den Panels aktualisiert.

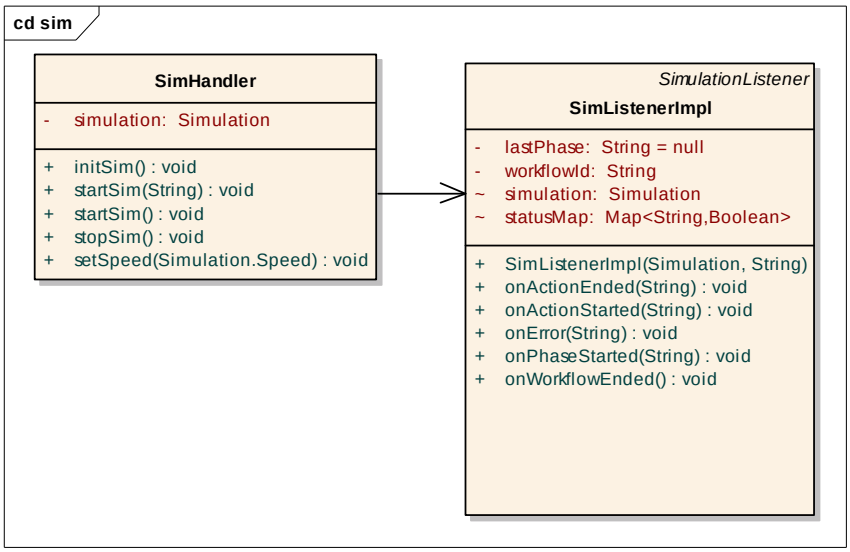

Abbildung 28: Simulation Klassen

4.6 Presentation Layer

4.6.1 Pakete

Der Presentation Layer beinhaltet alle Pakete die mit der grafischen Ausgabe zu tun haben. Dies sind die Panels und Dialoge sowie die Menus.

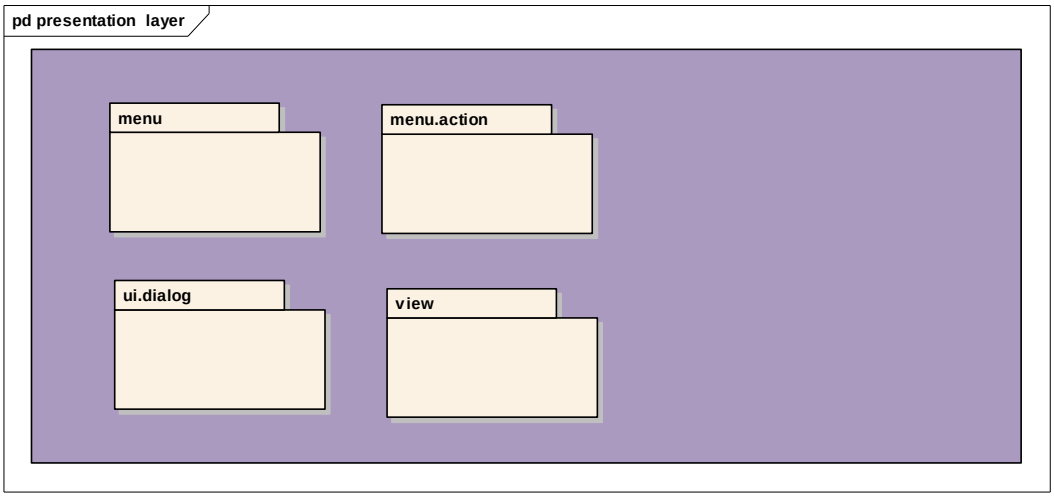

Abbildung 29: Paket Diagramm Presentation Layer

Tabelle 8: Übersicht Presentation Layer Pakete

Paket	Beschreibung
menu	Beinhaltet die Klassen für das Erstellen der Menus.
menu.action	Beinhaltet die Menu Action Klassen.
ui.dialog	Beinhaltet Dialog Fenster Klassen.
view	Beinhaltet die Panel Klassen.

4.6.2 Dialog

Für das Öffnen und Speichern eines Workflows werden zusätzliche Dialoge benötigt. Diese basieren auf den Standarddialogen. Es werden zusätzlich Filter für Filetypen und Selektionskriterien gesetzt. Die ShowDialogBox dient zum Anzeigen von Warnungen und Fehlermeldungen.

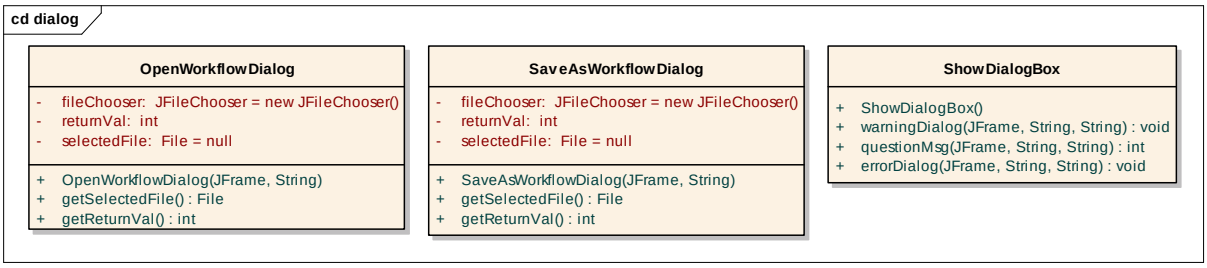

Abbildung 30: Dialog Klassen

4.6.3 Menu und menu.action

Die MenuBuilder Klasse baut das Menu auf. Dies geschieht mit Hilfe der jeweiligen MenuAction Klasse, z.B. OpenMenuAction, welche von der abstrakten MenuAction Klasse erben. Für jedes MenuItem gibt es eine entsprechende Klasse.

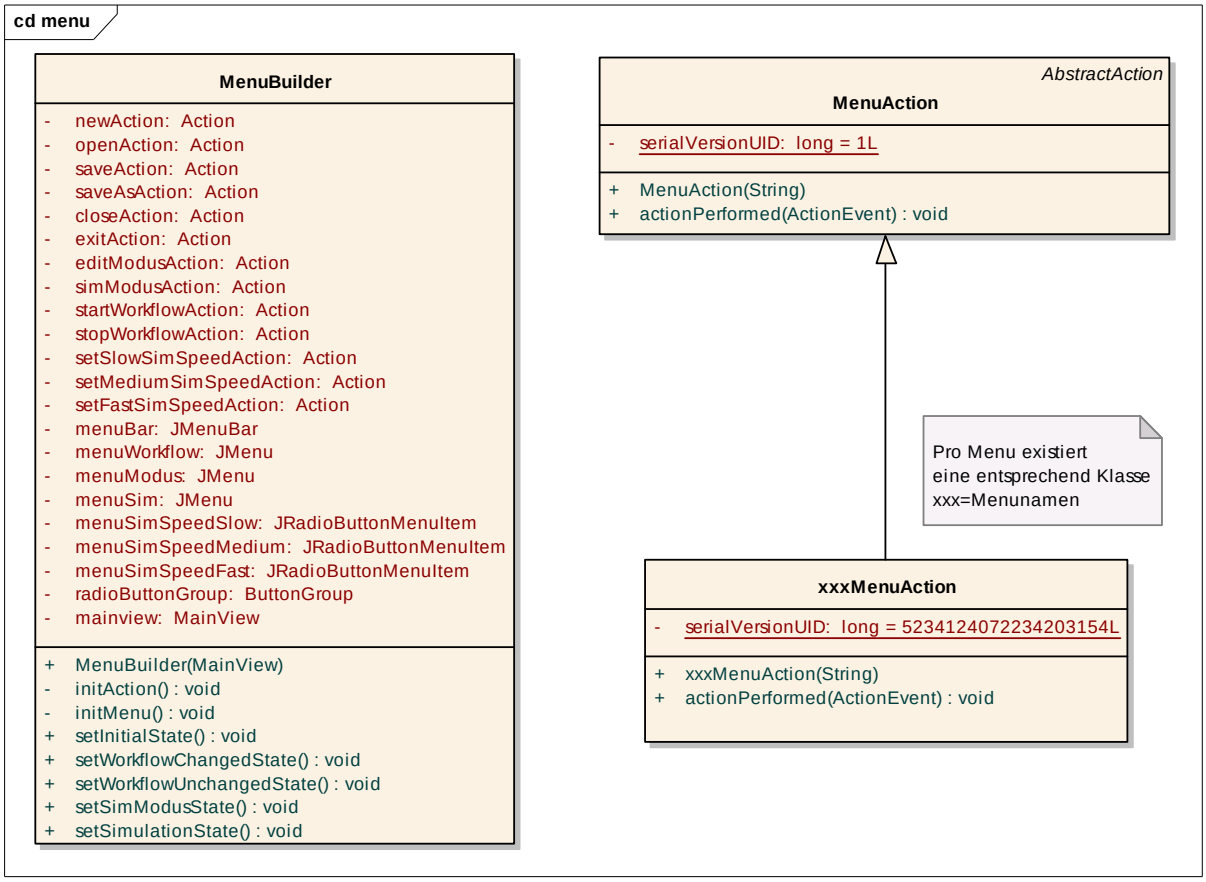

Abbildung 31: Menu und menu.action Klassen

4.6.4 View

Das View Paket beinhaltet die Mainview, welche alle anderen Klassen kennt. Das TreePanel ist verantwortlich für die Darstellung des Baumes mit den darin enthaltenen Elementen. Die auf dem Baum ausgeführten Aktionen werden ebenfalls direkt in dieser Klasse behandelt. Sämtliche anderen Panels sind für die Anzeige und Bearbeitung der Detailinformationen eines bestimmten Elements zuständig. Alle diese Panels wurden mit Hilfe eines GUI Editors erstellt und um die benötigte Funktionalität erweitert. Jedes Detailpanel hat eine setModel Methode mit welcher ihr das jeweilige Presentation Model übergeben wird. In diesem Model sind alle benötigen Daten für die Anzeige vorhanden. Zum Ändern der Daten muss auf dem erhaltenen Model die saveModel Methode mit den entsprechend veränderten Werten als Parameter aufgerufen werden. Im nachfolgenden Diagramm sind die Klassen des View Pakets zu sehen.

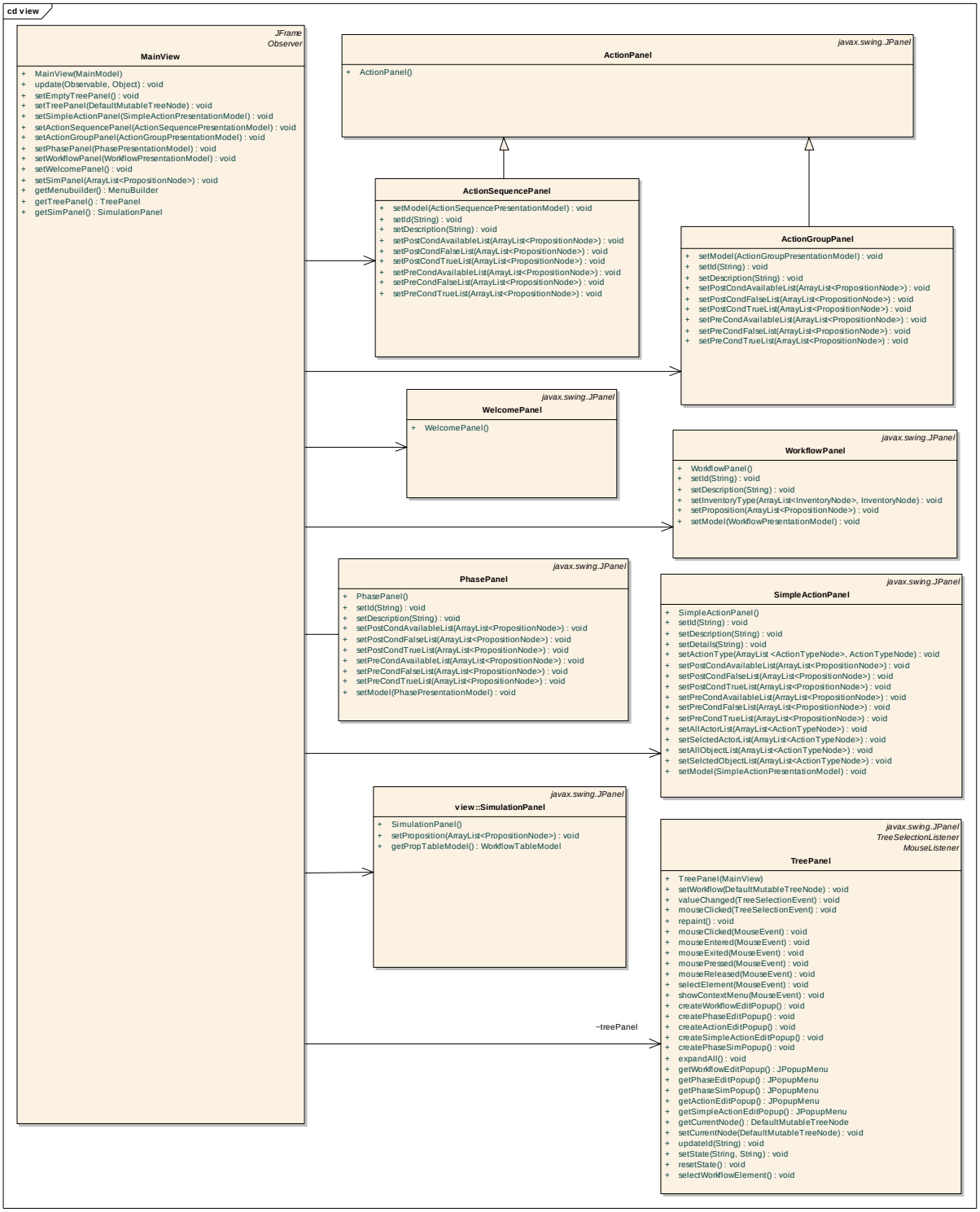

Abbildung 32: View Klassen (Panels)

4.7 Sequenz Diagramme

In diesem Kapitel soll mit Hilfe einiger ausgewählter Sequenz Diagramme auf bestimmte Abläufe genauer eingegangen werden.

4.7.1 Starten des HOVISSE Workflow Editor

Beim Starten des HOVISSE Workflow Editors wird als Erstes die Environment Klasse initialisiert. Dass heisst, es wird das hwfe.properties ausgelesen und geprüft ob es vorhanden ist und korrekt aussieht. Danach werden das MainModel und die MainView erstellt. Diese beiden Abläufe sind nachfolgend noch etwas detaillierter beschrieben. Anschliessend wird die MainView bei der TreeModel Klasse mit Hilfe des Observer Patterns registriert, damit diese bei einer Änderung des Models entsprechend informiert werden kann.
Als letzter Schritt wird der CommandHandler instanziiert und anschliessend initialisiert. Bei der Initialisierung werden ihm die Referenzen zum MainModel und der MainView mitgegeben.

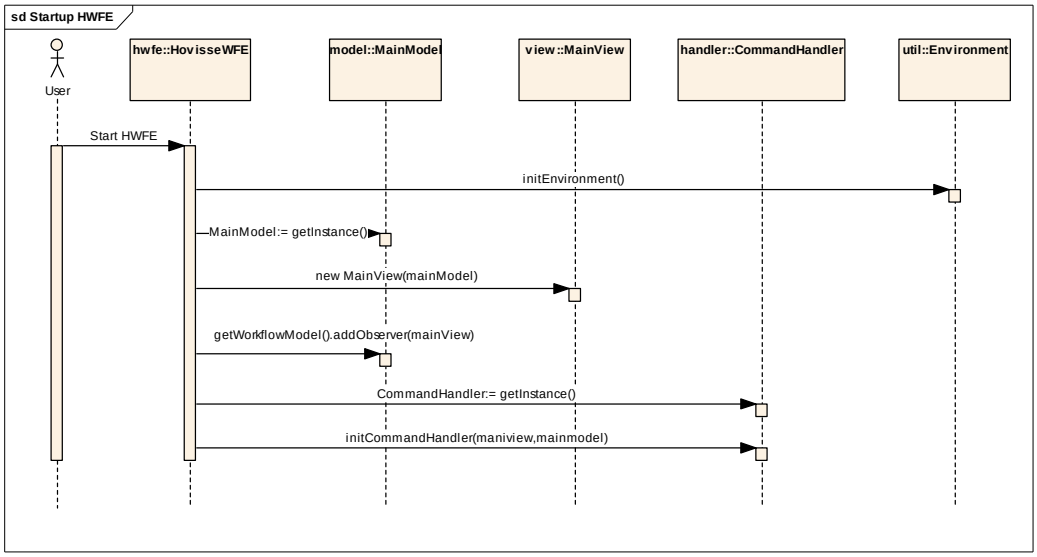

Abbildung 33: Sequenz Diagramm Startup

Beim Initialisieren des MainModels werden nacheinander die verschiedenen Models erzeugt, damit das MainModel die entsprechenden Referenzen besitzt.

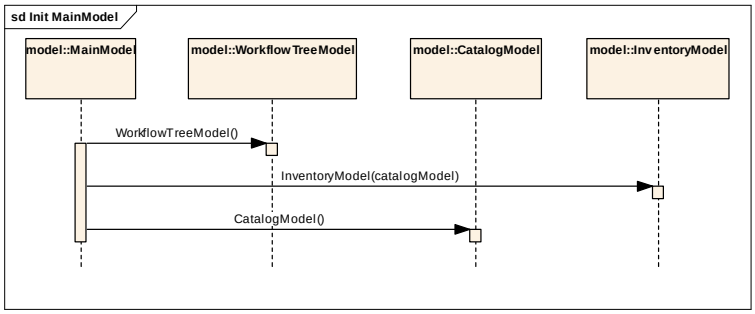

Abbildung 34: Sequenz Diagramm Init MainModel

4.7.1.1 Initialisieren der Main View (Init Main View)

Die Mainview selber ist das eigentliche Frame, da sie von JFrame erbt. Bei der Initialisierung der MainView wird als Erstes ein Menu mit Hilfe des MenuBuilders aufgebaut.
Für jeden Menupunkt ist eine entsprechende Action Klasse vorhanden, welche nun instanziiert wird. Zur besseren Übersicht ist nur ein solcher Ablauf im nachfolgenden Diagramm eingezeichnet. Anschliessend werden die eigentliche MenuBar und die Menus erstellt. Als Nächstes werden nun die Action Klassen welche die Menupunkte repräsentieren zu den entsprechenden Menus hinzugefügt. Als Letztes werden in den Menus die für den Initialzustand erlaubten Menupunkte aktiviert bzw. die anderen deaktiviert.

Da die Oberfläche in zwei Bereiche unterteilt werden soll, erstellt die initPanel Methode schliesslich ein SplitPanel und fügt es zur MainView hinzu.

Als letztes wird in den rechten Bereich des SplitPanels das Welcome Panel gesetzt. Der linke Bereich bleibt bis zum Vorhandensein eines Workflows leer.

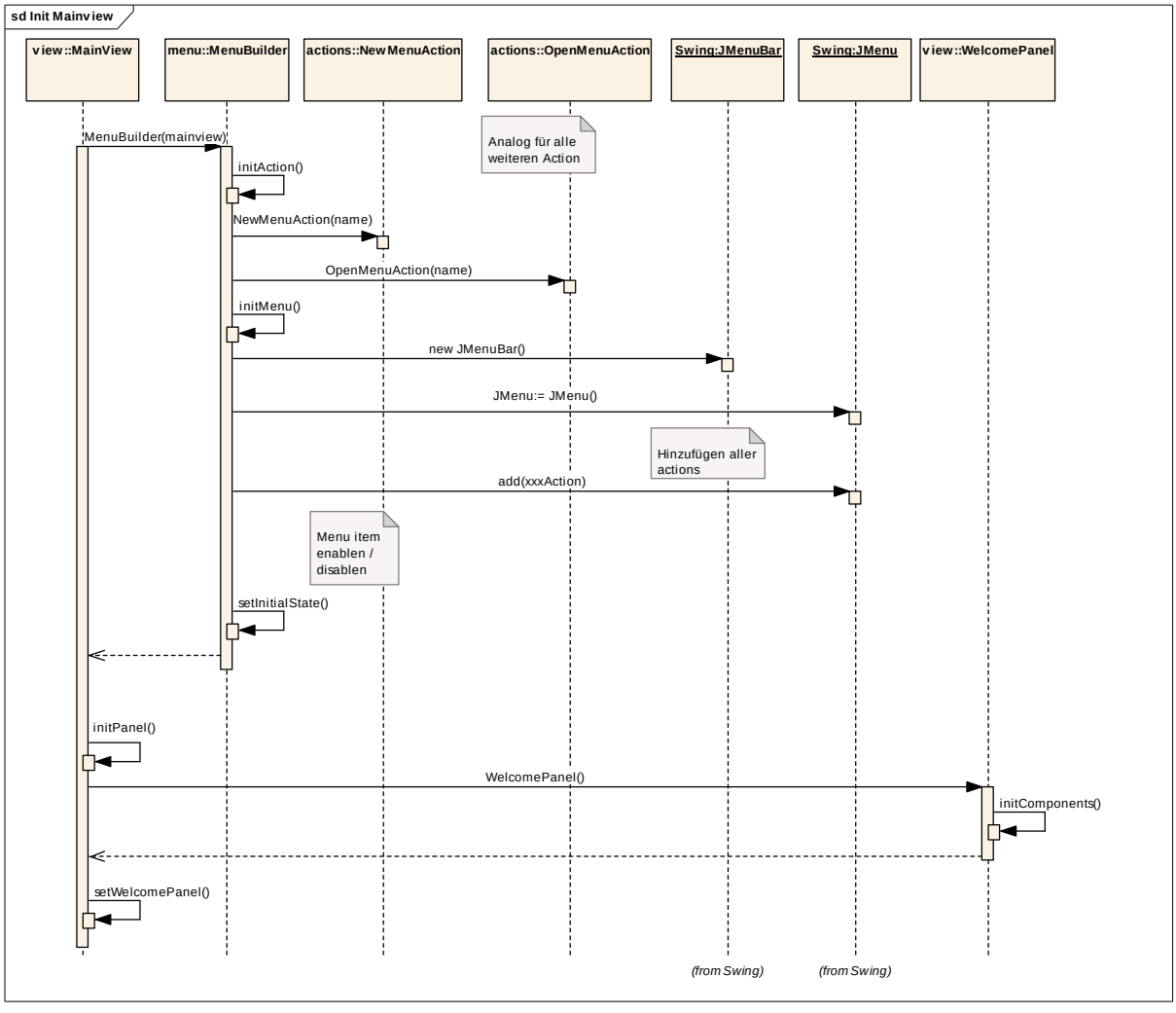

Abbildung 35: Sequenz Diagramm Init MainView

4.7.2 Öffnen des Workflow (Open Workflow)

Nach dem Starten des Programms ist das Öffnen eines bestehenden Workflows eine der möglichen nächsten Operationen.

Als Erstes wird ein OpenWorkflowDialog angezeigt mit welchem das einzulesende Workflow File ausgewählt werden kann. Der gewählte Dateipfad wird in der Environment Klasse abgelegt.
Mit Hilfe von JAXB wird das spezifizierte XML File eingelesen (deserialisert) und das entsprechende Objektmodel aufgebaut.
Mit der Methode CreateProposition werden die Proposition Objekte des Workflows in eine HashMap abgefüllt, da die anderen Elemente (Phase, Action usw.) darauf referenzieren. Somit muss bei einem Zugriff, was im Normalfall bei jedem Anzeigen einer Phase oder einer Action mehrmals der Fall ist, nicht immer die ganze Liste von Propositions durchsucht werden.
Als Nächstes wird mit Hilfe der createTree Methode der eigentliche Baum aufgebaut. Diese Methode ist nachfolgend noch detaillierter beschrieben.
Anschliessend wird nun der View mitgeteilt, dass ein neuer Baum vorhanden ist. Diese kann den neuen Baum somit entsprechende anzeigen.

Nun werden wiederum mit Hilfe von JAXB das Catalog File sowie das im Workflow referenzierte Inventory File eingelesen (deserialisiert) und die entsprechenden Objektmodele aufgebaut. Durch das Setzen des Catalog Objekts in das Model werden die einzelnen Elemente für einen schnelleren Zugriff wiederum in HashTables abgelegt. Das Gleiche gilt für die Inventory Objekte. Zusätzlich werden die vorhandenen Referenzen auf die Catalog Objekte innerhalb des Inventory Data Models aufgelöst. Wenn beim Lesen der Files ein Fehler auftritt oder die referenzierten Files nicht vollständig vorhanden sind, wird eine entsprechende Fehlermeldung ausgegeben und das Öffnen des Workflows abgebrochen.
Wenn der Workflow erfolgreich geladen werden konnte, wird im Titel der Pfad und Dateiname des geladenen Workflows angezeigt.

Nun wird ein neues WorkflowPresentationModel mit Hilfe des übergebenen Workflow Objekts erstellt. Das Model bereitet die Daten so auf, dass sie vom GUI möglichst einfach für die Darstellung übernommen werden können. Nun wird im rechten Teil des SplitPanels das Workflow Panel gesetzt. Für jeden Elementtyp gibt es nur eine Instanz, welche jeweils mit den aktuellen Werten gefüllt wird. Dies geschieht mit Hilfe des eben erstellten PresentationModel, welches als Parameter mitgeben wurde. Das PresentationModel wird im Gegensatz zum Panel für jedes gewählte Objekt immer wieder neu erstellt.
Als Letztes wird im Environment noch der neue Zustand mitgeteilt sowie die korrekten Menu Items aktiviert.

Der Ablauf für das Erstellen eines neuen Workflows ist sehr ähnlich. Anstelle des Einlesens eines Workflows wird mit Hilfe der generierten ObjectFactory eine neue Workflow Klasse erzeugt.

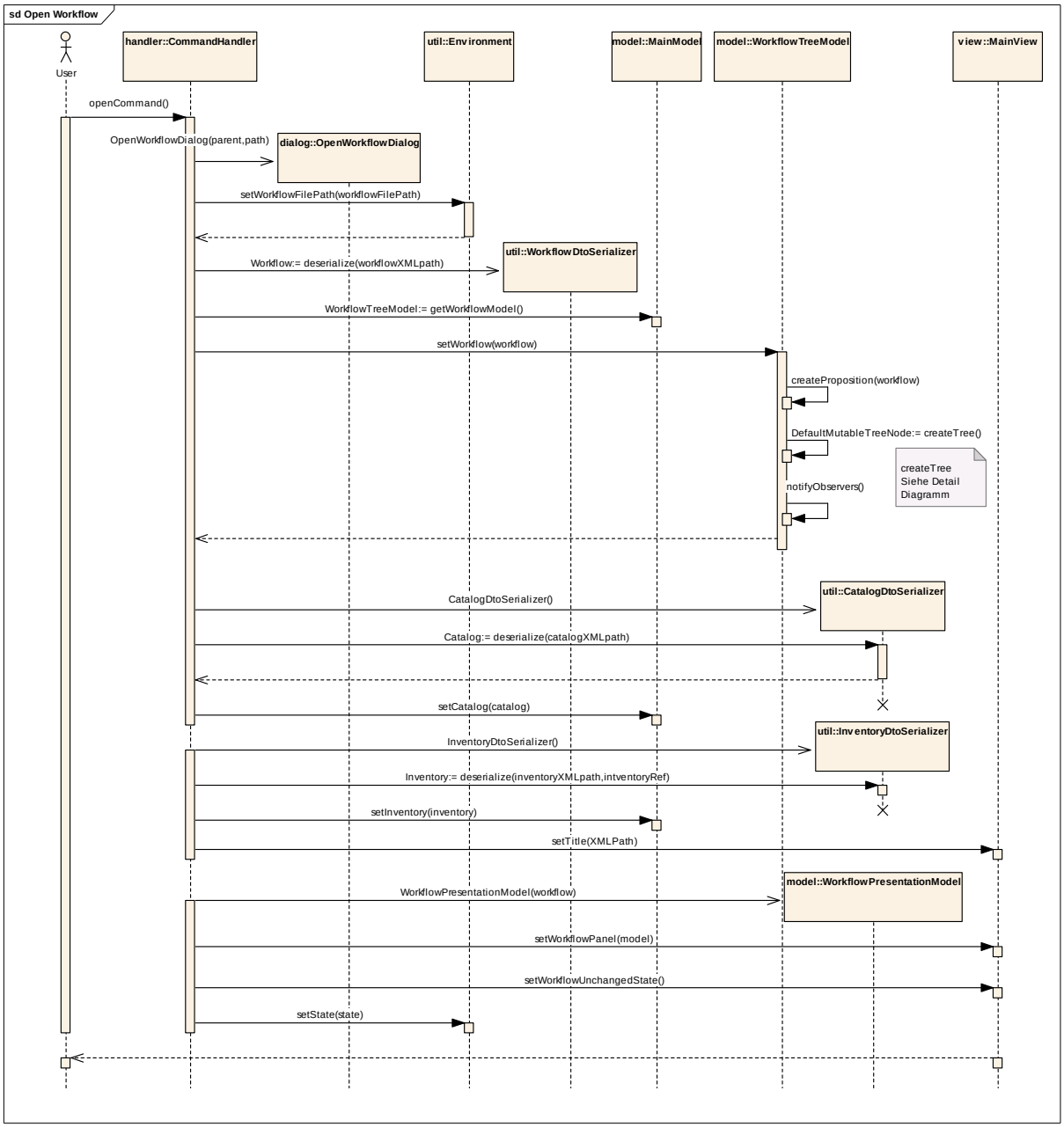

Abbildung 36: Sequenz Diagramm Open Workflow

4.7.2.1 Erzeugen des Baums (Create Tree)

Das Workflow Objekt weist im Prinzip bereits eine Art Baumstruktur auf. Somit muss das Workflow Objekt nur in einer pre-order Traversierung durchlaufen werden und dabei die entsprechenden Objekte erzeugt werden.
Der Baum wird dabei mit DefaultMutableTreeNode Objekten aufgebaut, welche ein Objekt enthalten können. Damit jedes dieser Objekte gleich behandelt werden kann, wurde ein einfaches Datenobjekt der TreeElementNode erstellt. Dieses enthält nebst dem eigenen Objekt noch einmal separat den Typ und den Namen (id). Zudem ist noch ein Status vorhanden, welcher bei der Simulation zu tragen kommt. Nebstdem ist die toString Methode wichtig, da sie zur Beschriftung des Elements im Baum verwendet wird.

Somit wird nun als Erstes ein neuer Node mit dem entsprechenden Element für den Workflow erstellt. Dann wird die erste Phase gesucht und entsprechend hinzugefügt. Anschliessend wird für alle Actions dieser Phase ein entsprechendes Objekt erstellt und an die aktuelle Phase angehängt. Dies wird dann für die nächsten Phasen wiederholt, bis der ganze Workflow durchlaufen wurde.

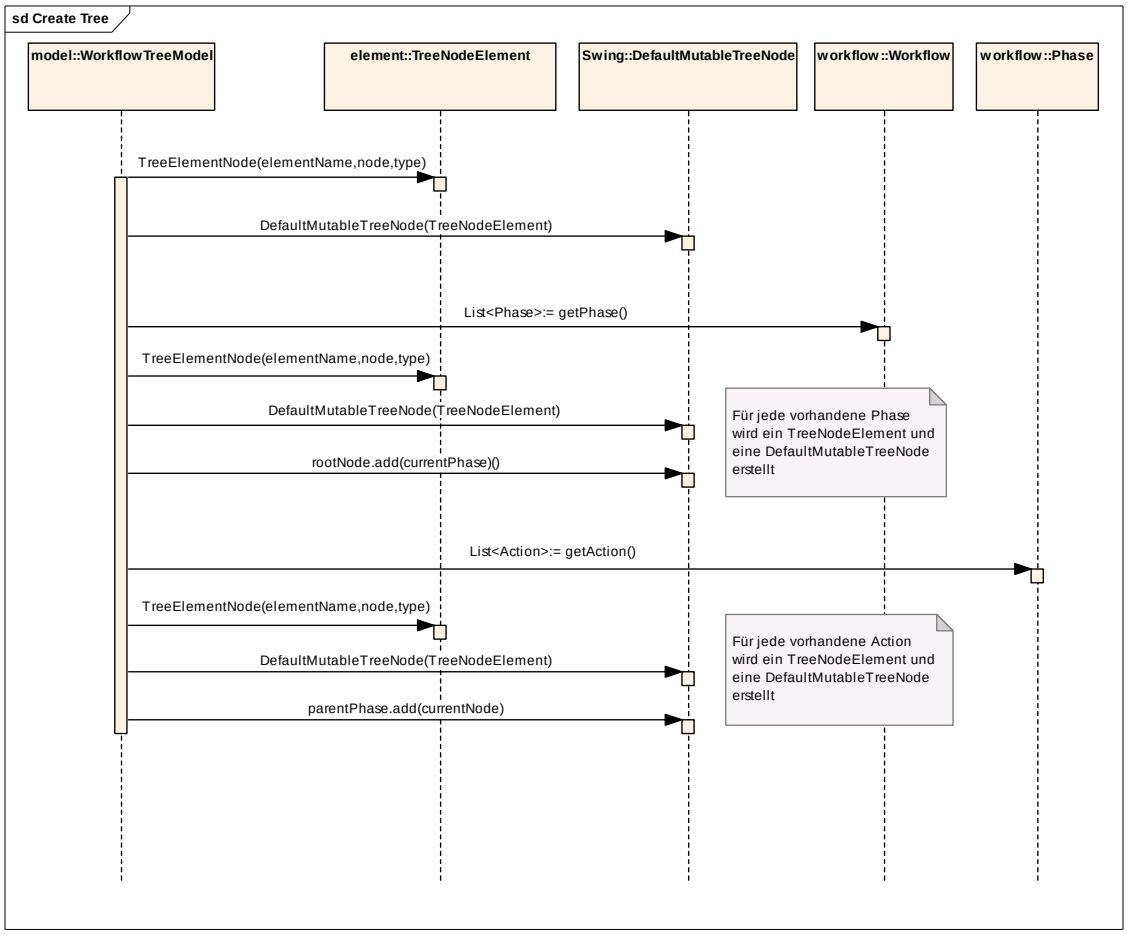

Abbildung 37: Sequenz Diagramm Create Tree

4.7.3 Selektieren eines Elements (Select Element)

Das Selektieren eines Elements ist ein wichtiger Schritt, welcher sich im Prinzip jedoch recht geschaltet.
Durch das Registrieren eines TreeSelectionListener wird beim Selektieren eines neuen Elements die Methode valueChanged aufgerufen, welche dann entsprechend implementiert werden muss.

Als Erstes muss die Referenz auf das angewählte Element geholt werden. Da im DefaultMutableTreeNode Objekt jeweils unser Daten Objekt enthalten ist, kann dies einfach herausgeholt und der setPanel Methode übergeben werden. Innerhalb dieser Methode wird nun entsprechend des enthaltenen Types das passende PresentationModel Objekt erzeugt, dem richtigen Panel übergeben und das Panel in die View gesetzt. Als Letztes wird das selektierte Objekt als das aktuelle Objekt gemerkt. Dies wird dann später wieder benötigt, wenn mit Hilfe des Kontext Menus eine Aktion darauf ausgeführt wird.

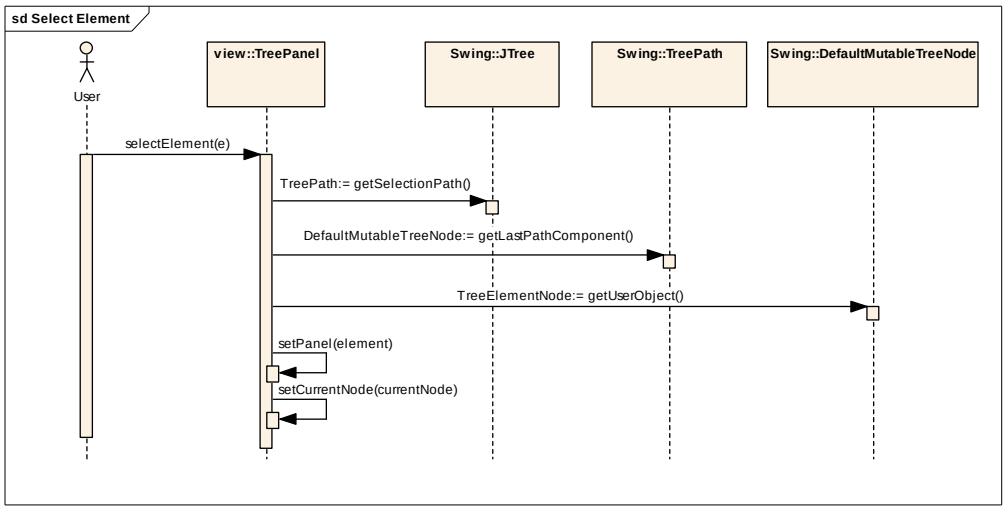

Abbildung 38: Sequenz Diagramm Select Element

4.7.4 Hinzufügen eines Elements (Add ActionGroup)

Das Hinzufügen eines Elements wird mit Hilfe des Beispiels einer ActionGroup erläutert. Grundsätzlich ist der Ablauf aber für alle Elemente sehr ähnlich.
Ausgelöst wird das Hinzufügen durch das Anwählen der entsprechenden User Interaktion (via. Rechtsklick auf eine Phase in Kontext Menu).
Als Erstes wird ein neues ActionGroup Objekt erzeugt und eine Default ID gesetzt. Es wird ein neues DefaultMutableTreeNode Element erzeugt und unser Datenobjekt hinzugefügt. Nun werden die Parent Objekte benötigt um die beiden Objekte (Daten und TreeNode) entsprechend hinzuzufügen. Da die entsprechende Operation bereits auf dem Parent Objekt ausgeführt wurde, (in unserem Fall Phase), ist dies entsprechend das aktuelle TreeNode Objekt bzw. das darin enthaltene Daten Objekt an welches unsere neu erzeugten Elemente nun hinzugefügt werden können.

Da nun das soeben hinzugefügte Element bearbeitet werden soll, muss als letzter Schritt noch der Fokus auf das entsprechende Element gesetzt werden und die Tree UI Komponente aktualisiert werden, damit das neue Element auch angezeigt wird.

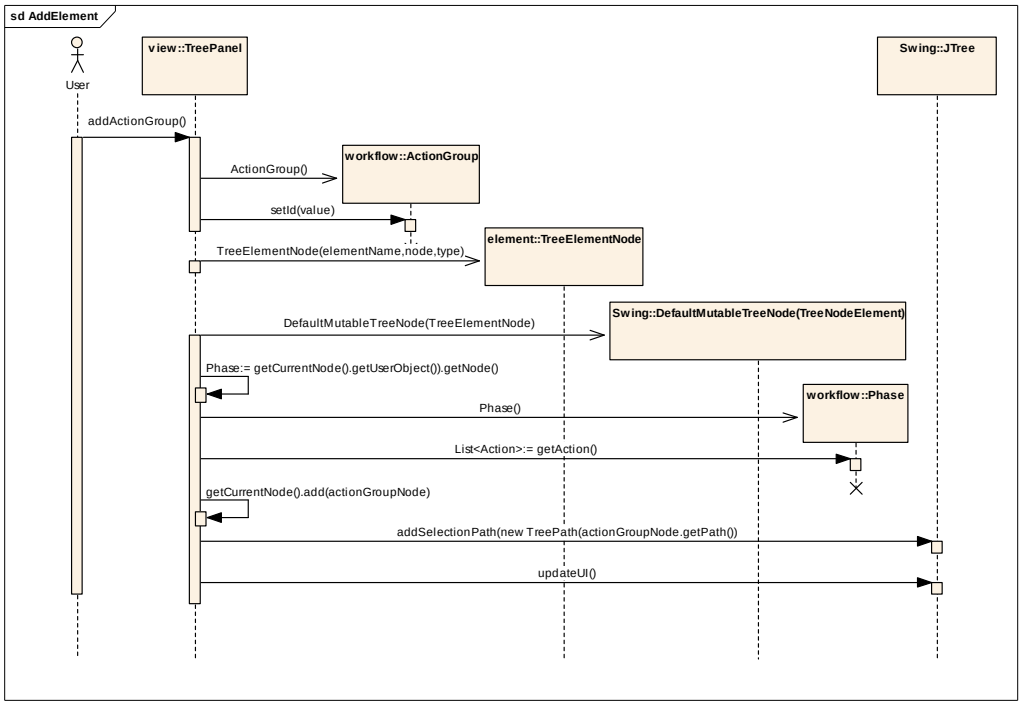

Abbildung 39: Add Element (ActionGroup)

5 Lösungsbeschreibung

Ein grosser Teil der Lösungsbeschreibung bzw. der Erkenntnisse sind bereits in den Kapiteln 4 Design und 3 Prototypen zu finden. Zudem ist im Kapitel 7.1 Usability Test die GUI Lösungsfindung beschrieben. In diesem Kapitel soll noch einmal auf gewisse Zusammenhänge und spezielle Gegebenheiten eingegangen werden.

5.1 Datenfluss

5.1.1 Erste Idee

Anfänglich bestand die Idee, wie im nachfolgenden Diagramm beschrieben, die aus den XML generierten Klassen (Workflow Objekte) direkt an das GUI zu übergeben und direkt mit den jeweiligen Elementen zu verbinden. Die im GUI gemachten Änderungen hätten dann jeweils in eine Kopie des XML Files geschrieben (serialisiert) werden können. Beim Verwerfen der Änderungen hätte dann nur das bestehende XML File neu geladen werden müssen.
Ein endgültiges Speichern hätte nur noch eine Umbenennung des Files nach sich gezogen.

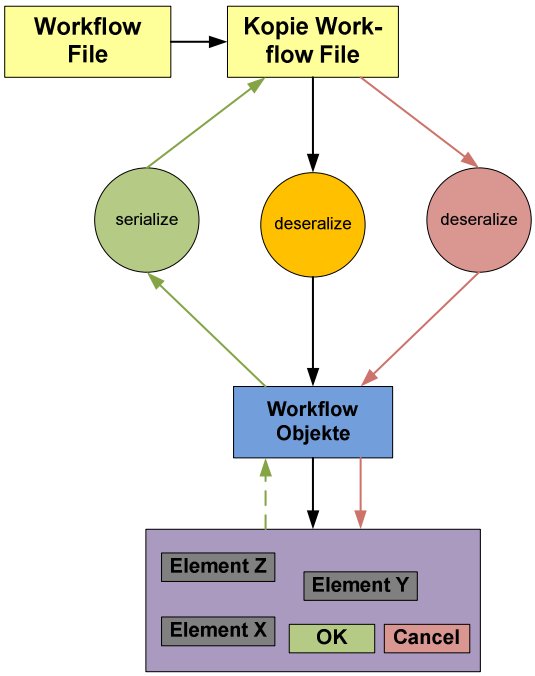

Abbildung 40: Erste Idee Datenfluss

Es hatte sich dann aber gezeigt, dass dies leider kein praktikabler Weg war. Dies vorwiegend, weil sich die Elemente des XML nicht so gut auf die GUI Elemente abbilden lassen. Es gibt viele Referenzen, einerseits bereits innerhalb des Workflow XML Files und andererseits auf die Inventory und Catalog Files.

5.1.2 Umgesetzter Datenfluss

Die Data Files (Workflow, Catalog und Inventory) werden mit Hilfe von JAXB auf Java Objekte gemappt. Diese werden innerhalb der entsprechenden Data Model Klassen abgelegt. Zusätzlich werden bereits erste Abhängigkeiten aufgelöst und die Daten für den Zugriff optimiert, das heisst die Daten welche mit der zur Verfügung gestellten Getter Methoden abgefragt werden, sollen einfach zugreifbar sein. So werden zum Beispiel Listen für einen schnelleren Zugriff in Hashtables abgefüllt.
Pro anzuzeigendes Element (Phase p2, Action a4, Action a5, usw.) wird jedes Mal ein Presentation Model für das gewählte Element zu der Zeit an welcher es benötigt wird, wieder neu erstellt. Das Presentation Model hält dabei nur Referenzen auf die vorhandenen Objekte in den Data Models. Die Presentation Models haben dabei die Aufgabe die Daten in einer für die View Panel optimierten Form mit Hilfe von Getter Methoden pro zu anzeigendes Element zur Verfügung zu stellen. Das Presentation-Model wird dann an das entsprechende Panel übergeben, welches die Werte zur Anzeige bringt.

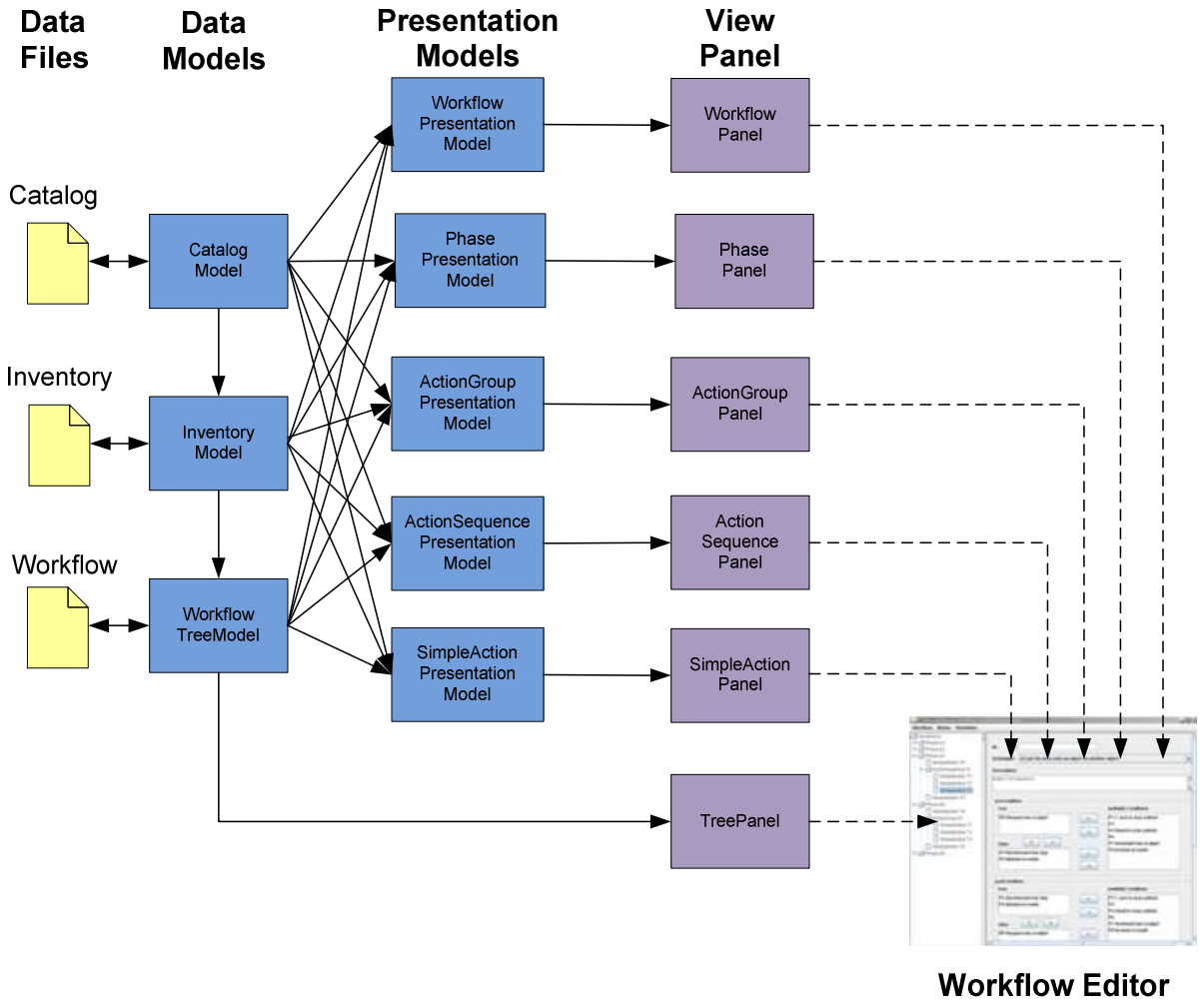

Abbildung 41: Übersicht Datenfluss

5.2 Abweichungen zum Pflichtenheft

Ursprünglich war in der Konfiguration ein definition.path vorgesehen in welchem sich die Schemas und dtd Files befinden. In JAXB wird das dtd File standardmässig im gleichen Verzeichnis wie das XML File erwartet. Ich habe keine Möglichkeit gefunden dies einfach anzupassen. Deshalb wurde der definition.path weggelassen. Die dtd und Schema Files müssen sich nun in den jeweils entsprechenden Foldern der XML Files befinden.

5.3 Nützliche Hinweise

In diesem Kapitel habe ich einige nützliche Codeteile aufgelistet, für Probleme die im ersten Moment einige Schwierigkeiten bereitet haben, aber mit ganz wenig Code gelöst werden konnten.

5.3.1 Schliessen mit dem X Button

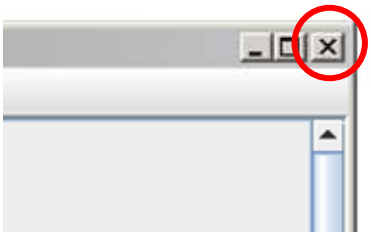

Abbildung 42: Schliessen mit dem X Button

Standardmässig ist der X Button oben rechts (roter Kreis) bereits so vorbelegt, dass beim Anwählen die Applikation automatisch geschlossen wird.
Falls nun aber eigener Code ausgeführt werden soll, zum Beispiel um vorgängig zu überprüfen, ob alles gespeichert wurde und allenfalls eine Warnung auszugeben, muss die Funktion überschrieben werden.

Dazu muss einerseits ein neuer addWindowsListener zum eigenen JFrame hinzugefügt werden und die windowsClosing Methode überschrieben werden.
In meinem Fall habe ich einfach mein bestehendes exitCommand aufgerufen.
Zum Anderen musste definiert werden, dass die DefaultCloseOperation nichts tut.

```
myJFrame.addWindowListener(new java.awt.event.WindowAdapter(){
      public void windowClosing(java.awt.event.WindowEvent evt)
            {
                  // persönlicher Exit Handler aufrufen
                  commandHandler.exitCommand();

            }
});

/* Disable automatic close operation */
myJFrame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

Code 1: Schliessen mit dem X Button

5.3.2 Ausfüllen eines Panels mit dem Tree Panel

Für den Editor wurde ein Split Panel benutzt, in welches auf der linken Seite jeweils mit Hilfe eines JTree der Workflow dargestellt wird. Dabei ist anfänglich das Problem aufgetreten, dass das JTreePanel nicht den ganzen vorhandenen Platz ausgefüllt hat (linke Abbildung). Durch den untenstehenden Code, in welchem vorgängig ein GridLayout der Grösse 1 definiert wird, kann das Problem behoben werden und das JTreePanel füllt danach jeweils den ganzen vorhandenen Platz (rechte Abbildung) aus.

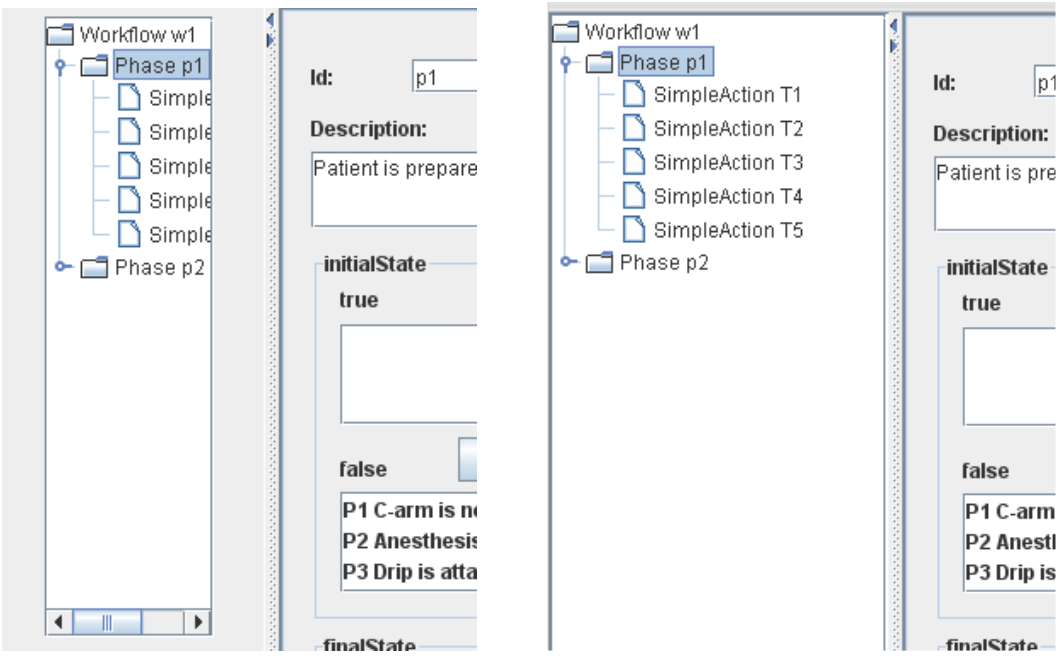

Abbildung 43: Ausfüllen eine Panels

```
public TreePanel(MainView mainview ) {
      super (new GridLayout(1,1));
      ...
}
```

Code 2: Ausfüllen eine Panels

6 Tools & Technologien

Nachfolgend soll ein Überblick über die eingesetzten Tools und Technologien gegeben werden.

6.1 Entwicklungsumgebung

Als Entwicklungsumgebung wurde Eclipse Version 3.3 eingesetzt. Diese Entscheidung beruht auf die in den letzten Jahren damit gesammelten guten Erfahrungen.

Der Mangel in Eclipse ist ein fehlender GUI Editor. Ein solcher ist mit Matisse in Netbeans der zweiten weit verbreiteten freien Entwicklungsumgebung vorhanden.
Entgegen der vorgesehenen Idee das GUI dort zu pflegen, wurde der Editor nur zur einmaligen Erzeugung der entsprechenden Panels verwendet und diese dann in die Eclipse Entwicklungsumgebung übernommen und entsprechend erweitert.

Wenn die Erweiterung sauber vom generierten Code getrennt wird, können im Bedarfsfall einfach wieder neue Panel Klassen generiert und die eigenen Teile entsprechend hinzugefügt werden.

6.2 Weitere Tools

6.2.1 Subversion (SVN)

Für die Verwaltung und das Versionshandling wurde ein Subversion Repository eingesetzt. So war jederzeit gewährleistet, dass gemachte Änderungen nachvollzogen und diese im Bedarfsfall wieder rückgängig gemacht werden konnten.

Weitere Informationen zu SVN sind unter http://subversion.tigris.org zu finden.

6.2.2 UML

Für die UML Diagramme wurden das Tool Enterprise Architect (EA) eingesetzt, da ich bereits in mehreren Projekten gute Erfahrungen damit gesammelt hatte und sich die Diagramme relativ gut exportieren lassen.

Enterprise Architect ist ein leistungsfähiges UML Tool, welches im Normalfall allen Ansprüchen für ein durchschnittlich komplexes Entwicklungsprojekt genügt. Es ist relativ einfach zu bedienen und wird im Gegensatz zu manch anderem Tool zu einem vernünftigen Preis angeboten.

Weitere Informationen zu EA gibt es unter http://www.sparxsystems.de/.

6.3 Swing

Swing ist ein Framework zur Erstellung von grafischen Benutzeroberflächen und ist im Gegensatz zu SWT (Standard Widget Toolkit) Plattform unabhängig, das heisst es werden keine grafischen Komponenten des Betriebssystems verwendet. Bereits seit der Java Version 1.2 ist es ein fester Bestandteil der Java Runtime Environment (JRE).

6.4 JAXB

Die Idee von JAXB (Java Architecture for XML) ist die Anbindungen von XML Files an Java Klassen und somit eine einfache Manipulation von XML Files zu ermöglichen. JAXB erzeugt aus dem XML Schema ein Abbild in Java Klassen auf welche entsprechend zugegriffen werden kann. Das Lesen (Unmarshalling) des XML Files in die Klassen bzw. das Zurückschreiben (Marshalling) wird von JAXB durchgeführt. Somit sind keine Kenntnisse über die Bearbeitung von XLM Dateien mehr notwendig. JAXB kann als Ersatz von SAX und DOM angesehen werden. JAXB Version 2.0 ist Bestandteil der Java SE 6.

Weitere Informationen zu JAXB sind unter https://jaxb.dev.java.net/ und http://java.sun.com/developer/technicalArticles/WebServices/jaxb/ zu finden.

6.5 Log4j

Ausser der Dokumentation ist auch das Fehlerhandling eine wichtige Komponente des Entwicklungszyklus. Nebst einer vereinfachten Fehlerausgabe soll für die genauere Untersuchung der Fehler die entsprechenden Exceptions geloggt werden können. Zudem kann so bei Bedarf zusätzliche Debug Information geloggt werden um den Verlauf und Zustand einer Applikation während der Laufzeit auszugeben bzw. gespeichert werden, um sie einer späteren Analyse zu unterziehen.

Für das Logging wird das Standard Logframework Log4J eingesetzt. Es hat sich in den letzten Jahren sozusagen als de facto Standard etabliert.

Weitere Informationen zu Log4j sind unter http://logging.apache.org/log4j/ zu finden.

7 Test

7.1 Usability Test

Es war das Ziel mit Hilfe der Usabilty Test ein GUI zu entwerfen, welches einerseits logisch aufgebaut ist, die Daten übersichtlich darstellt und eine hohe Qualität bezüglich der Anwendbarkeit und Benutzerfreundlichkeit aufweist. Ein weiterer Anspruch war es, das GUI so zu gestalten, dass der User möglichst wenig fehlerhafte Eingaben tätigen kann.

Die durchgeführten Usability Tests sind in diesem Fall mehr als eine Art laufender Prozess anzusehen. Es wurden mehrmals neue Dialoge diskutiert. Zum Teil nur mit Hilfe von Skizzen auf Papier, zum Teil mit Hilfe von GUI Prototypen ohne vollständige Funktionalität, welche mit einem GUI Editor erstellt worden waren. Dabei wurden die Panels zum Teil mehrmals überarbeitet, bis schliesslich der heutige Stand erreicht wurde. Nachfolgend ist der Ablauf dieser Tests beschrieben.

Ausgangspunkt waren die GUI Entwürfe aus dem Pflichtenheft. Dabei wurden folgende Dinge festgestellt:

- Die id muss ebenfalls bearbeitet werden können.
- Der ActionType muss verändert und dabei aus einer vorgegebenen Menge ausgewählt werden können.
- Die Inventory Referenz gehört nur zum Workflow.
- Es müssen actor und object dargestellt werden können.
- Die Auswahl / Bearbeitung der Proposition ist zu kompliziert.

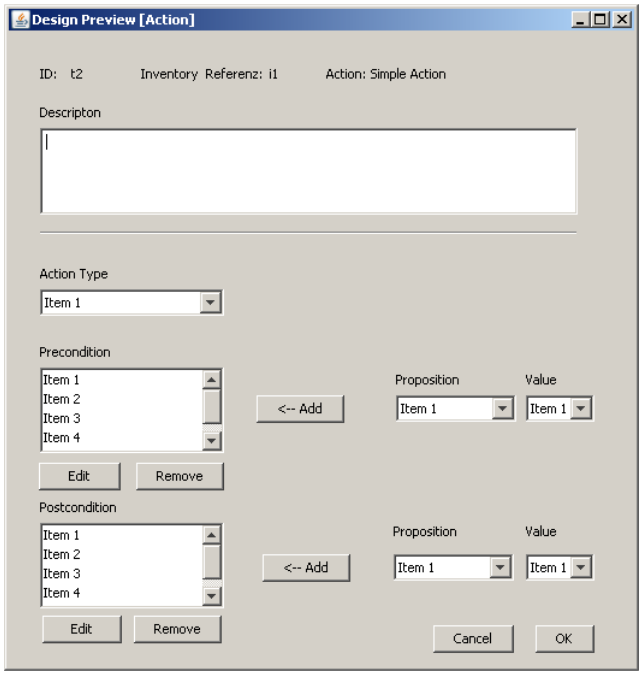

Abbildung 44: Erster Panel Entwurf für eine Action (Aus Pflichtenheft)

7.1.1 actor und object

Für die actor und object wurde festgestellt, dass diese zum Teil eine grosse Menge von zusätzlichen Informationen enthalten können, welche nicht auf dem bestehenden Panel Platz finden. Die Informationen werden aber teilweise benötigt um entscheiden zu können, ob und welche actor und object zu einer Action hinzugefügt werden sollen.
Eine erste Idee war ein Detail Button, der für ein bestimmtes Element ein Detail Dialog mit den zusätzlichen Informationen öffnet.
Bei einem weiteren Review wurde aber festgestellt, dass ein zusätzlicher Dialog eher unpraktisch ist und den Arbeitsablauf stört. Bei der Suche nach einer Lösung bin ich auf die Idee gekommen, die Detailanzeige direkt in das Panel zu integrieren und dort jeweils die Details zum letzten selektierten actor oder object Element anzuzeigen. Siehe dazu im Kapitel 9.3 Selektieren / Bearbeiten von Elementen die Abbildung 48: Details zu selektiertem Element.

7.1.2 Proposition

Die ursprünglichen Dialoge haben sich als zu komplex erwiesen, bzw. wurden bei einem ersten GUI Review als zu „unpraktisch“ befunden. Bei einer zu grossen Menge von Proposition ist eine ComboBox nicht mehr praktisch. Für die Bearbeitung wäre zudem ein zusätzlicher Dialog nötig geworden. Das Löschen schliesslich, wäre über das Selektieren eines Elements durchgeführt worden. Das heisst, für jede der drei Operationen wäre eine Handlung nötig gewesen.
Aufgrund dessen, habe ich mich ein wenig umgeschaut, wie dies in ähnlichen Fällen gemacht wird. Häufig wird dabei ein Dialog mit 2 Listboxen verwendet, wobei eine Seite die verfügbaren und die andere Seite die ausgewählten Elemente enthält.
Ein solcher Dialog steht in Java nicht zur Verfügung, weshalb habe ich einen entsprechenden Dialog, wie nachfolgend abgebildet erstellt habe.

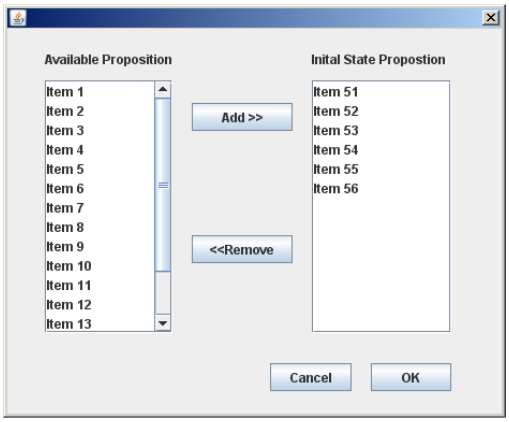

Abbildung 45: 2-Fach Auswahlbox (Prototype)

Dabei war die Idee, mit Hilfe der 2-Fach Auswahlbox jeweils die gewünschten Propositions auszuwählen. Dies hat jedoch zu verschiedenen neuen Problemen geführt. Die Zuweisung des Wertes (true oder false) der Proposition findet nicht bei der Auswahl statt. Somit hätte ein default Wert gesetzt werden müssen oder der

Werte wäre bis zur Definition leer geblieben. Dies hätte in einem Fall dazu geführt, das man sich die neu hinzugefügten Elemente hätte merken müssen, da sie nicht als neue Werte erkennbar gewesen wären. Andererseits hätte bei leeren Werten die Gefahr bestanden, dass diese nie auf einen entsprechend gültigen Wert gesetzt worden wäre. Dies hat dann dazu geführt, dass ich auf die Idee mit dem 3-Fachen Auswahlpanel gekommen bin.

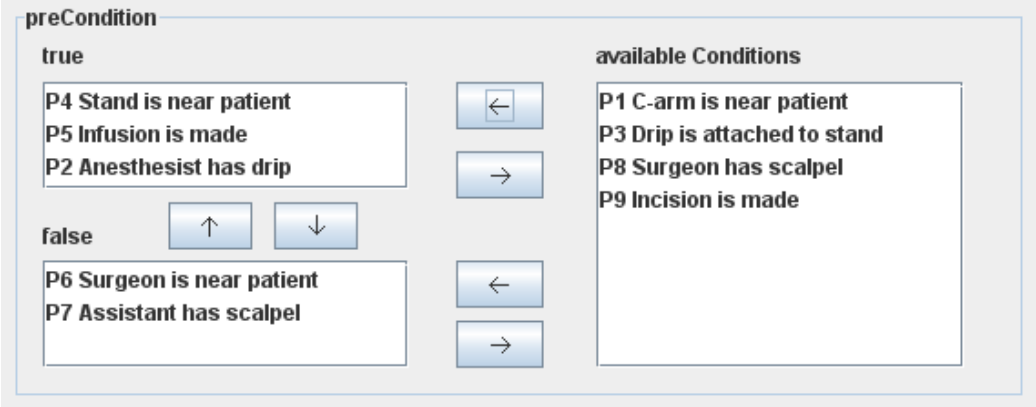

Abbildung 46: Ausschnitt 3-Fach Auswahlbox (Definitive Lösung)

Dort kann der Anwender die entsprechenden Werte sehr einfach zur gewünschten Gruppe hinzufügen bzw. von dieser entfernen. Für alle Operationen (Hinzufügen, Löschen, Ändern) wird immer die gleiche Operation durchgeführt. In einer ersten Idee sollte das 3-Fache Auswahlpanel noch in einem separaten Fenster geöffnet werden. Dies hatte jedoch wiederum den Nachteil eines zusätzlichen Fensters das jedesmal geöffnet und wieder geschlossen werden muss und somit den Arbeitsfluss bremst. Dies auch weil die Arbeitsweise eher so ist, dass wenn ich für eine Vorbedingung einen bestimmten Wert setze, ich gleichzeitig die Nachbedingung für den entsprechenden Wert hinzufüge. Deshalb wurde dies wieder verworfen und sämtliche Funktionen in das jeweilige Panel eingebaut. So sind immer alle definierten und möglichen Werte auf einen Blick erkennbar. Des Weiteren sind die True und False Bedingungen getrennt, was meiner Ansicht nach zu einer besseren Übersicht beiträgt.

7.2 System Test Cases

Im Pflichtenheft wurde spezifiziert, dass für alle definierten Use Cases Testfälle durchzuführen sind.
Da gewisse Use Cases an mehreren Stellen verwendet werden und noch zusätzliche Tests erfolgt sind, war eine eins zu eins Relation zu den Use Cases nicht sinnvoll.
In der nachfolgenden Tabelle ist jeweils spezifiziert mit welchem Test Case (TC) jeweils welche Use Cases (UC) getestet wurden.

Tabelle 9: Übersicht der Test Cases

Test Case Nummer	Test Case Bezeichnung	Getestete Use Cases
#1.1	Workflow Editor starten	-
#1.2	Workflow Editor beenden	-
#1.3	Neuer Workflow erstellen	UC #1.1, #2.5.1.1, #2.5.1.2, #2.5.1.3, #2.7
#1.4	Bestehender Workflow öffnen	UC #1.2
# 1.5	Bestehender Workflow speichern	UC #1.3, #1.4
# 1.6	Workflow schliessen	UC #1.5
#1.7	Workflow bearbeiten	UC #1.2, #1.3, #2.2, #2.2.1, #2.2.2, #2.2.3, #2.5, #2.5.1, #2.5.1.1, #2.5.1.2, #2.5.1.3, #2.8
#1.8	Elemente hinzufügen	UC #2.1, # 2.1.1, #2.1.2, #2.1.3, #2.4, #2.7
#1.9	Elemente entfernen	UC #2.3, #2.6
# 1.10	Modus & Simulation	UC #1.6, #1.7, #3.1, #3.1.1, #3.2, #3.3

7.3 Test Cases

Tabelle 10: Workflow Editor starten #TC 1.1

Ausgangslage		Der Workflow Editor ist korrekt konfiguriert.
Aktion		Der Worklow Editor wird gestartet.
Test ID		
1.1.1	Linkes Panel	▪ Das Panel ist minimiert (stellt nichts dar).
1.1.2	Rechtes Panel	▪ Das Panel zeigt das „Begrüssungs“ Panel an.
1.1.3	Menu	▪ Nur die Menu Items New, Open und Exit sind aktiv.
Aktion		
Test ID		
1.1.4	Rechtes Panel	Das Panel zeigt wieder die ursprünglichen Werte an.
Ausgangslage		Das hwfe.properties ist nicht vorhanden.
Aktion		Der Workflow Editor wird gestartet.
Test ID		
1.1.5	Dialog	▪ Es wird eine Fehlermeldung angezeigt. ▪ Das Programm wird beendet.

Tabelle 11: Workflow Editor beenden #TC 1.2

Ausgangslage		Der Workflow Editor ist gestartet und ein Workflow ist verändert.
Aktion		Es wird im Menu Workflow Exit angewählt.
Test ID		
1.2.1	Linkes Panel	▪ Es wird eine Warnung angezeigt, dass der Workflow verändert aber nicht gespeichert wurde.
Aktion		Es wird Nein angewählt.
Test ID		
1.2.2	Rechtes Panel	▪ Das Beenden wird abgebrochen. Der Editor ist wieder im Zustand wie vor dem Anwählen von Exit.
Aktion		Es wird im Menu Workflow Exit angewählt. Bei der Warnung wird Ja angewählt.
Test ID		
1.2.3	Dialog	▪ Das Programm wird ohne zu speichern beendet.

Tabelle 12: Neuer Workflow erstellen #TC 1.3

Ausgangslage		Der Editor ist korrekt konfiguriert und gestartet.
Aktion		Es wird im Menu Workflow New angewählt.
Test ID		
1.3.1	Linkes Panel	▪ Das Panel zeigt den Baum mit einem Workflow Tree Element an.
1.3.2	Rechtes Panel	▪ Das Panel zeigt die leere Workflow Dialog Maske an. ▪ Es können Werte für id und description gesetzt werden. ▪ Es können Proposition hinzugefügt, geändert und entfernt werden. ▪ Es kann ein Inventory ausgewählt werden.
1.3.3	Menu	▪ Die Menu Items Save, SaveAs, Close und Exit sind aktiv.
Aktion		Es werden mehrere Phasen hinzugefügt und Werte für die entsprechenden Elemente gesetzt.
Test ID		
1.3.4	Linkes Panel	▪ Das Panel zeigt die hinzugefügten Phasen an.
1.3.5	Rechtes Panel	▪ Das Panel zeigt jeweils die Phase Dialog Maske mit den definierten Werten an.
Aktion		Der Workflow wird gespeichert.
Test ID		
1.3.6	XML File	▪ Der erstellte Workflow und die entsprechend definierten Werte sind im gespeicherten File vorhanden.

Tabelle 13: Bestehender Workflow öffnen #TC 1.4

Ausgangslage		Der Editor ist korrekt konfiguriert und gestartet.
Aktion		Es wird im Menu Workflow Open angewählt. Es wird kein File ausgewählt und cancel gewählt.
Test ID		
1.4.1	Editor	▪ Das rechte Panel zeigt weiterhin das „Begrüssungs“ Panel.
Aktion		Es wird im Menu Workflow Open angewählt. Es wird ein ungültiges oder unkorrektes File ausgewählt.
Test ID		
1.4.2	Dialog	▪ Es wird eine Fehlermeldung angezeigt. Der Editor bleibt im initialen Zustand.
1.4.3	Menu	▪ Es sind weiterhin nur die Menu Items New, Open und Exit aktiv.
Aktion		Es wird im Menu Workflow Open angewählt. Es wird ein korrektes File ausgewählt.
Test ID		
1.4.4	Linkes Panel	▪ Es wird der Workflow als Tree dargestellt.
1.4.5	Rechtes Panel	▪ Es werden die Details zu den Workflow Elementen angezeigt.
1.4.6	Menu	▪ Es sind die Menu Items SaveAs, Close und Exit aktiv.
1.4.7	Titel	▪ Es wird das aktuell geladene File mit Pfad angezeigt.

Tabelle 14: Bestehender Workflow speichern #TC 1.5

Ausgangslage		Es ist ein Workflow geöffnet, der verändert wurde.
Aktion		Es wird im Menu Workflow Save angewählt.
Test ID		
1.5.1	XML File	▪ Der veränderte Workflow wir in das bestehende File geschrieben.
1.5.2	XML File	▪ Das bisher bestehende „alte“ File wird unter <Filename> <DatumZeit>.bak gesichert.
1.5.3	Menu	▪ Das Menu Item „Save“ ist nicht mehr aktiv.
Aktion		Es wird im Menu Workflow Save As angewählt und ein Filename ausgewählt der bereits vorhanden ist.
Test ID		
1.5.4	XML File	▪ Der geladene Workflow wird in das gewählte File gespeichert.
1.5.5	Menu	▪ Das bisher bestehende „alte“ File wird unter <Filename> <DatumZeit>.bak gesichert.
1.5.6	Titel	▪ Es wird das neue File mit Pfad angezeigt.
Aktion		Es wird im Menu Workflow Save As angewählt und ein neuer Filename definiert.
Test ID		
1.5.7	XML File	▪ Der geladene Workflow wird in das gewählte File gespeichert.
1.5.8	Titel	▪ Es wird das neue File mit Pfad angezeigt.

Tabelle 15: Workflow schliessen #TC 1.6

Ausgangslage		Es ist ein Workflow geöffnet, der verändert wurde.
Aktion		Es wird im Menu Workflow Close angewählt.
Test ID		
1.6.1	Dialog	▪ Es wird eine Warnung angezeigt, dass der Workflow verändert aber noch nicht gespeichert wurde.
Aktion		Es wird Nein angewählt.
Test ID		
1.6.2	Editor	▪ Das Schliessen wird abgebrochen. Der Editor ist wieder im Zustand wie vor dem Anwählen von Close.
Aktion		Es wird im Menu Workflow Close angewählt. Bei der Warnung wird Ja gewählt.
Test ID		
1.6.3	Linkes Panel	▪ Das Panel ist minimiert (stellt nichts dar).
1.6.4	Rechtes Panel	▪ Das Panel zeigt wieder das „Begrüssungs“ Panel.
1.6.5	Menu	▪ Die Menu Items New, Open und Exit sind aktiv
1.6.6	XML	▪ Es wurde nichts gespeichert. Die gemachten Änderungen sind verloren.

Tabelle 16: Workflow bearbeiten #TC1.7

Ausgangslage		Es ist Workflow geöffnet. Es wurde ein Action Element im Baum angewählt.
Aktion		Es werden id und/oder description verändert, sowie die Zustände pre – und postCondition angepasst (True / False ändern, hinzufügen und entfernen). Es wird cancel gedrückt.
Test ID		
1.7.1	Rechtes Panel	▪ Die Werte werden wieder auf ihre ursprünglichen Werte zurückgesetzt.
Aktion		Es werden id und/oder description verändert, sowie die Zustände pre – und postCondition angepasst (True / False ändern, hinzufügen und entfernen). Es wird OK gedrückt.
Test ID		
1.7.2	Rechtes Panel	▪ Die neuen Werte sind im Model gespeichert. ▪ Die veränderten Werte erscheinen auch wieder nach einem zwischenzeitlichen Anwählen eines anderen Elements.
Aktion		Es werden im Workflow zwei neue Propositionen hinzugefügt. Es wird die description einer Proposition geändert. Es wird eine Phase angewählt.
Test ID		
1.7.3	Rechtes Panel	▪ Die neuen Werte sind in der Phase verfügbar.

Aktion		Die zwei neuen Propositionen werden als True oder False Condition hinzugefügt. Eine bestehende Condition wird entfernt. Es wird im Menu Save oder SaveAs gewählt.
Test ID		
1.7.4	XML	▪ Das File wird geschrieben. ▪ Im gewählten XML File sind die Werte wie vorgängig definiert vorhanden, bzw. angepasst.

Tabelle 17: Elemente hinzufügen #TC1.8

Ausgangslage		Es ist ein Workflow geöffnet.
Aktion		Es werden neue Propositionen zum Workflow hinzugefügt.
Test ID		
1.8.1	Rechtes Panel	▪ Die neuen Propositions sind in den Phasen unter den „available States“ bzw. in den Conditions unter „available Conditions“ vorhanden.
Aktion		Es wird eine neue Phase hinzugefügt. Es werden entsprechende Werte spezifiziert.
Test ID		
1.8.2	Linkes Panel	▪ Die neu erstellte Phase wurde zum Baum hinzugefügt.
Aktion		Es wird eine neue ActionGroup zu einer Phase hinzugefügt. Es werden mehrere SimpleActions zur ActionGroup hinzugefügt.
Test ID		
1.8.3	Linkes Panel	▪ Die neu erstellten Actions wurden zum Baum hinzugefügt.
Aktion		Es wird eine neue ActionSequence zu einer Phase hinzugefügt. Es werden mehrere SimpleAction zur ActionSequence hinzugefügt.
Test ID		
1.8.4	Linkes Panel	▪ Die neu erstellten Actions wurden zum Baum hinzugefügt.
Aktion		Der aktuelle Workflow wird gespeichert.
Test ID		
1.8.5	XML	▪ Das File wird geschrieben. ▪ Im gewählten XML File sind die Werte wie vorgängig angepasst vorhanden.

Tabelle 18: Elemente entfernen #TC1.9

Ausgangslage		Es ist ein Workflow geöffnet der mehrere Elemente von jedem Type aufweist. Der Tree ist vollständig geöffnet.
Aktion		Es wird eine Phase entfernt, an welcher mehrere Action angehängt sind.
Test ID		
1.9.1	Linkes Panel	▪ Die Phase inklusive aller zugehörigen Subelemente wurde aus dem Baum entfernt.
1.9.2	Rechtes Panel	▪ Es werden die Details zum Parent Element des gelöschten Elements angezeigt.
Aktion		Es wird eine ActionGroup entfernt, an welcher mehrere SimpleAction angehängt sind.
Test ID		
1.9.3	Linkes Panel	▪ Die ActionGroup inklusive aller zugehörigen Subelemente wurde aus dem Baum entfernt.
1.9.4	Rechtes Panel	▪ Es werden die Details zum Parent Element des gelöschten Elements angezeigt.
Aktion		Es wird eine ActionSequence entfernt, an welcher mehrere SimpleAction angehängt sind.
Test ID		
1.9.5	Linkes Panel	▪ Die ActionSequence inklusive aller zugehörigen Subelemente wurden aus dem Baum entfernt.
1.9.6	Rechtes Panel	▪ Es werden die Details zum Parent Objekts des gelöschten Objekts angezeigt.
Aktion		Es werden nacheinander mehrere Action entfernt.
Test ID		
1.9.7	Linkes Panel	▪ Die Actions wurden aus dem Baum entfernt.
1.9.8	Rechtes Panel	▪ Es werden die Details zum Parent Elementen des gelöschten Objekts angezeigt.
Aktion		Der aktuelle Workflow wird gespeichert.
Test ID		
1.9.9	XML	▪ Das File wird geschrieben. ▪ Im gewählten XML File sind die gelöschten Elemente nicht mehr vorhanden.

Tabelle 19: Modus & Simulation #TC 1.10

Ausgangslage		Ein Workflow ist geladen und ist unverändert (gespeichert).
Aktion		Es wird im Menu Modus SimulationModus angewählt.
Test ID		
1.10.1	Linkes Panel	▪ Der Tree wird expandiert. ▪ Ein Anwählen eines Elements hat keine Wirkungen mehr. ▪ Es ist nur noch das Kontext Menu Start Phase verfügbar.
1.10.2	Rechtes Panel	▪ Es werden die Propositions angezeigt.
1.10.3	Menu	▪ Das StartWorkflow Menu ist aktiv. ▪ Das StopWorkflow Menu ist inaktiv. ▪ Die Geschwindigkeit Menus können angewählt werden.
Aktion		Die Geschwindigkeit wird auf „FAST“ gesetzt. Es wird eine Phase gestartet.
Test ID		
1.10.4	Linkes Panel	▪ Die Tree Elemente werden laufend entsprechend ihrer aktuellen Zustände eingefärbt.
1.10.5	Rechtes Panel	▪ Es werden laufend die Zustände der Proposition entsprechend ihrem aktuellen Zustand angezeigt / verändert.
Aktion		Die Geschwindigkeit wird auf „SLOW“ gesetzt. Die gleiche Phase wird erneut gestartet.
Test ID		
1.10.6	Linkes Panel	▪ Die Abarbeitung dauert länger als bei der letzten Simulation.
Aktion		Es wird im Menu Modus EditModus angewählt.
Test ID		
1.10.7	Menu	▪ Die Simulation Menu Items sind wieder deaktiviert.
1.10.8	Linkes Panel	▪ Der Tree wird wieder „normal“ (ohne Einfärbung) dargestellt. ▪ Die Kontext Menus sind wieder verfügbar.
1.10.9	Rechtes Panel	▪ Es wird das Workflow Detail Panel dargestellt.
Aktion		Es wird im Menu Modus SimulationModus angewählt. Es wird Workflow starten angewählt.
Test ID		
1.10.10	Linkes Panel	▪ Die Tree Elemente werden laufend entsprechend ihrer aktuellen Zustände eingefärbt. ▪ Das Kontext Menu Start Phase ist nicht mehr verfügbar.
1.10.11	Rechtes Panel	▪ Es werden laufend die Zustände der Propositions entsprechend ihrem aktuellen Zustand angezeigt / verändert.
1.10.12	Menu	▪ Das StartWorkflow Menu ist inaktiv. ▪ Das StopWorkflow Menu ist aktiv.
Aktion		Es wird Workflow stoppen angewählt.
Test ID		
1.10.13	Linkes Panel	▪ Der letzte Zustand bleibt bestehen.
1.10.14	Rechtes Panel	▪ Der letzte Zustand der Proposition wird angezeigt.

Da sämtliche Tests erfolgreich durchgeführt wurden, wird auf eine Test Resultat Liste verzichtet.

8 Fazit

8.1 Vorgehen / Persönlicher Eindruck

Ein wichtiger Grundstein für die vorliegende Arbeit war sicherlich das Pflichtenheft. Ich habe es für die Erweiterung der Analyse und des Design, bzw. für die Implementierung doch des Öfteren wieder zur Hand genommen um zu prüfen wie etwas spezifiziert war.

Als speziell aufwendig habe ich die GUI Element erachtet. Diese werden wohl des Öfteren unterschätzt und da es dort nicht ein richtig oder falsch gibt, können dort fast beliebig lange Optimierungen vorgenommen werden. Ein Problem waren z.B. die optionalen Werte wie die Beschreibung bei den Properties. Somit müsste bei der Anzeige derselben eigentlich immer die nichtssagende ID dargestellt werden. Da ich dies ein wenig unschön fand, habe ich mich zu einer Kombination von ID und Description entschlossen.

8.2 Resultat

Ich denke, es konnte ein brauchbares Tool erstellt werden, dass die gestellten Anforderungen erfüllen kann. Sämtliche vorgesehenen Use Cases konnten umgesetzt werden und wurden erfolgreich getestet.

Ein negativer Punkt ist die aufgewendet Zeit. So hat die ganze Arbeit doch einiges mehr an Zeit in Anspruch genommen, als die ursprünglich vorgesehenen 360 Stunden.

8.3 Ausblick

Da die Zeit sehr knapp war, habe ich bewusst auf die Optimierung der optischen Details verzichtet und mich stattdessen primär auf die Funktionalität konzentriert.
So könnten im Tree eigene Icons verwendet werden. Das GUI könnte mittels Short Keys benutzbar gemacht werden oder um eine Toolbar erweitert werden.
Des Weiteren könnte mit relativ geringem Aufwand das Inventory File und allenfalls das Catalog File editierbar gemacht werden.
Eine Erweiterung mit der Workflow Engine wäre eine Einzelschritt Simulation.

9 Benutzerhandbuch

Dieses Kapitel soll in einer ganz kurzen Form die wichtigsten Aspekte aus Benutzersicht beschreiben. Dies sind einerseits die Konfiguration und andererseits die Bedienung des Workflow Editors selber. Dieser sollte jedoch möglichst selbsterklärend aufgebaut sein.

9.1 Konfiguration Files

9.1.1 hwfe.properties

Im hwfe.properties müssen die Pfade zu den benötigten Files definiert werden. Es muss im root Verzeichnis des Workfloweditors vorhanden sein.

Im catalog.path und inventory.path müssen sich das catalog.xml bzw. die benötigten inventory.xml befinden, sowie die jeweils zugehörigen dtd Files. In diesen beiden Verzeichnissen dürfen sich keine andern .xml Files befinden. Im definition.path sind die Schema Files abzulegen. Der workflow.path definiert das Verzeichnis in welchem sich die Workflow Files befinden.

```
catalog.path = c:/hwe/workspace/catalog
definition.path = c:/hwe/workspace/definition
inventory.path = c:/hwe/workspace/inventory
workflow.path = c:/hwe/workspace/workflow
```

Code 3: Beispiel hwfe.properties

9.1.2 log4j.properties

Im log4j.properties kann der Pfad zum Logfile und das Loglevel definiert werden.
Es muss sich ebenfalls im root Verzeichnis des Workfloweditors befinden.
Die möglichen Einstellungen sind unter
http://logging.apache.org/log4j/1.2/manual.html erklärt.

```
log4j.rootLogger=DEBUG, root

log4j.appender.root=org.apache.log4j.RollingFileAppender
log4j.appender.root.File=hwfe.log
log4j.appender.root.MaxFileSize=10MB
log4j.appender.root.MaxBackupIndex=10
log4j.appender.root.layout=org.apache.log4j.PatternLayout
log4j.appender.root.layout.ConversionPattern=%d{ISO8601}  %-5p  [%t]
%c: %m%n
```

Code 4: Beispiel log4.properties

9.2 Laden / Öffnen / Speichern eines Workflows

Über das Menu kann ein neuer Workflow erstellt oder ein bestehender Workflow (der Definition entsprechendes XML) geöffnet bzw. nach der Bearbeitung wieder gespeichert werden.

9.3 Selektieren / Bearbeiten von Elementen

Der Workflow wird auf der linken Seite als Baumstruktur aufgebaut. Elemente welche weitere Subelemente enthalten, können geöffnet werden, um diese Subelemente darzustellen. Auf der rechten Seite werden die Details zu dem angewählten Element dargestellt. Diese können dort entsprechend geändert werden.
Für die Auswahl der States und Conditions gibt es einen sogenannten 3-Fach Chooser. Die Werte können also zwischen true, false und available (nicht ausgewählt) hin und her geschoben werden.
Durch Auswählen von OK werden die geänderten Werte zwischengespeichert. Durch das Auswählen von Cancel oder Anwählen eines anderen Elements im Baum werden die gemachten Änderungen für das aktuelle Element verworfen bzw. auf den letzten Stand zurückgesetzt.

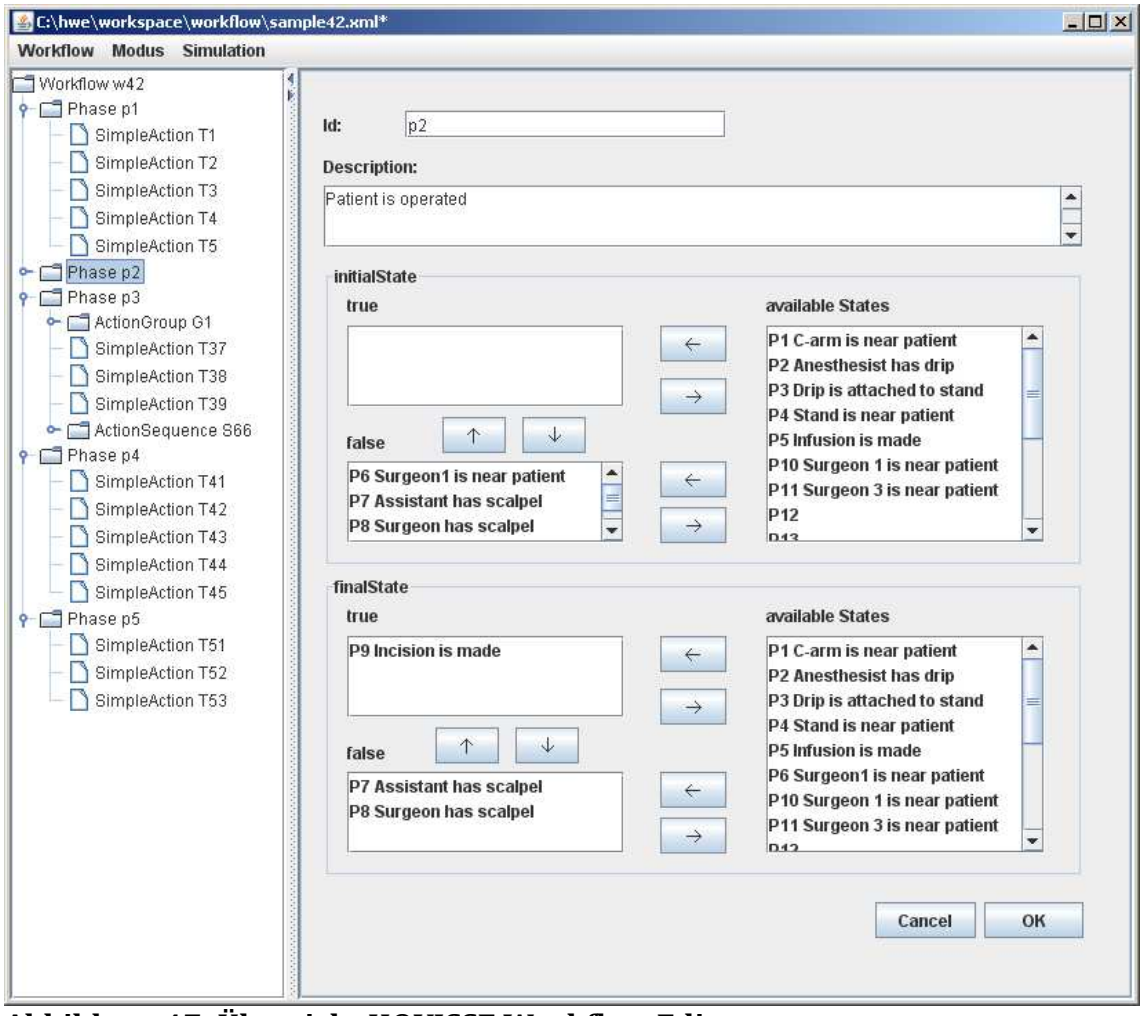

Abbildung 47: Übersicht HOVISSE Workflow Editor

Da die Elemente object und actor Zusatzinformationen enthalten können, welche aus Platzgründen nicht beim eigentlichen Element dargestellt werden können, gibt es ganz unten ein Detail Panel. Dort werden diese Details für das jeweils zuletzt selektierte actor oder object Element angezeigt.

Abbildung 48: Details zu selektiertem Element

9.4 Hinzufügen / Entfernen von Tree Elementen

Durch einen Rechtsklick auf ein Element in Baum können je nach Elementtyp (Phase, Action usw.) weitere Subelemente hinzugefügt oder das angewählte Element gelöscht werden. Beim Löschen eines Elements werden, falls vorhanden, alle zugehörigen Subelemente ebenfalls gelöscht.

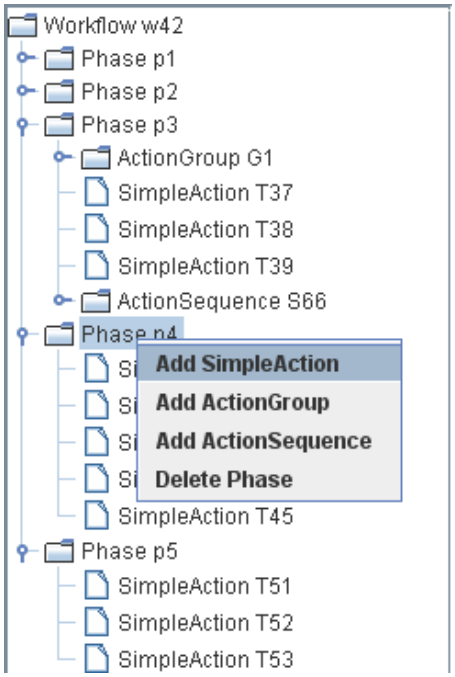

Abbildung 49: Bearbeiten des Workflow Trees

9.5 Simulation Modus / Edit Modus

Wenn ein Workflow gespeichert ist, kann im Menu Modus der Simulationsmodus angewählt werden. Nach einer abgelaufenen oder gestoppten Simulation kann wieder in den Edit Modus zurück gewechselt werden.

9.6 Simulation Menu

Nach einer Umschaltung in den Simulation Modus werden die entsprechenden Menu Punkte freigeschaltet. Die Simulation kann gestartet und es die gewünschte Simulationsgeschwindigkeit gesetzt werden.

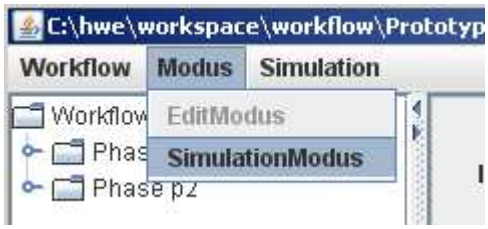

Abbildung 50: Menu SimulationsModus

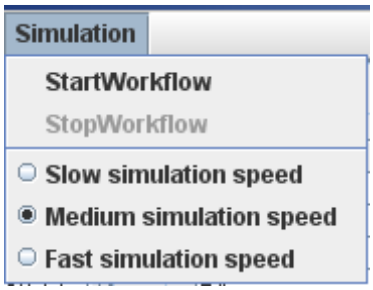

Abbildung 51: Menu Simulation

9.7 Simulation

Nach dem Start einer Simulation werden links im Baum die Elemente entsprechend ihrem Zustand eingefärbt. Gelb bedeutet, dass eine Phase oder Action gestartet wurde. Grün bedeutet, dass die entsprechende Phase oder Action erfolgreich beendet wurde. Wenn der ganze Workflow erfolgreich durchlaufen wurde, wird das Workflow Element ebenfalls grün eingefärbt.

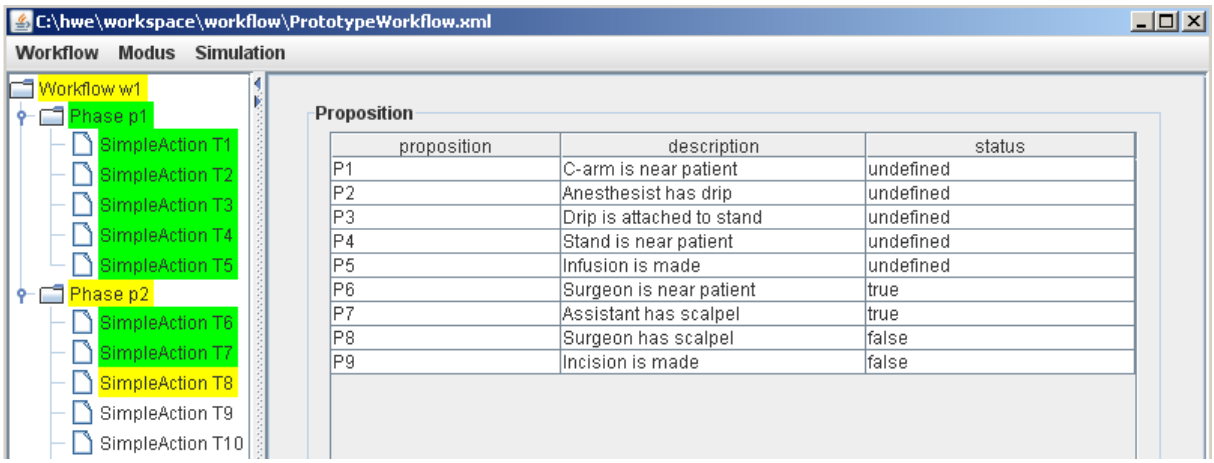

Abbildung 52: Workflow Editor während einer Simulation

A. XML Schemas & Files

Für die Datenhaltung und den Austausch mit der Workflow Engine werden XML Files benutzt, welche einem XML Schema genügen sollen.

Diese sind ausführlich im Pflichtenheft [2] beschrieben.

B. Glossar

HOVISSE	Haptic Osteosynthesis Virtual Intraoperative Surgery Support Environment.
Look and Feel	Bezeichnet das Aussehen und die Handhabung von Dialogen.
RUP	Rational Unified Prozess.
Swing	Ist eine Java Bibliothek zum Erstellen von grafischen Benutzer-oberflächen.
UML	Unified Modeling Language.
Virtual Reality	Bezeichnet computergenerierte künstliche Umgebungen.
XML	eXtensible Markup Language.
JAXB	steht für Java Architecture for XML Binding. Es ermöglicht aus einem XML Schema Java Klassen zu generieren und diese an ein entsprechendes File zu binden.

C. Referenzen

[01] Amrhein, B., & Fischli, S. (2007). Diplomthema: HOVISSE Workflow Editor. Bern.

[02] Baumeler, E, (2008). Pflichtenheft: HOVISSE Workflow Editor. Bern.

[03] Andreas Spillner, Tilo Linz. (2005). Basiswissen Softwaretest. Heidelberg: dpunkt.verlag.

[04] Computer Perception and Virtual Reality Group. (4. Januar 2007). Abgerufen am 13. April 2008 von http://www.cpvr.ch/?Virtual_Reality_Projects:ORISOS

[05] Flückiger, M. (2006). Usability Engineering. Schlieren (Zürich): Zühlke Engineering.

[06] Grau, R. (2006). Module Testing und Test Management. Schlieren (Zürich): Zühlke Engineering.

[07] Happel, J. (2006). Requirements Management. Schlieren (Zürich): Zühlke Engineering.

[08] Kruchten, P. (1999). Der Rational Unified Process. München: Addison-Wesley-Longman.

[09] Metz Igor;Scheidegger Andreas. (2003). OO Softwareentwicklung mit der UML. Bern: Glue Software Engineering.

[10] Oestereich, B. (2006). Analyse und Design mit UML 2.1. München: Oldenbourg Verlag.

[11] Oestereich, B. (2005). Die UML Kurzreferenz. München: Oldenbourg Verlag.

[12] Osley, M. (2003). Business Modeling based on RUP. Schlieren (Zürich): Zühlke Engineering.

[13] Scheuring Heinz; Scholian Thomas. (2001). Fachbuch Projektmanagement. Kaiseraugst: Scheuring AG.

[14] Kim Topley, (2000) Core Swing advanced programming, New Jersey, Prentice Hall

[15] Sun Swing Tutorial, https://java.sun.com/docs/books/tutorial/uiswing/index.html

[16] JAXB Reference Implementation, https://jaxb.dev.java.net/

D. Tabellenverzeichnis

Tabelle 1: Übersicht der möglichen Menupunkte pro Zustand 10
Tabelle 2: Übersicht der angezeigten Kontext Menu Punkte pro Zustand 10
Tabelle 3: Übersicht der Prototype Tests mit JAXB 17
Tabelle 4: Übersicht der Prototype Tests für JTree 19
Tabelle 5: Übersicht Support Layer Pakete 23
Tabelle 6: Übersicht Data Layer Pakete 25
Tabelle 7: Übersicht Presentation Layer Pakete 31
Tabelle 8: Übersicht Presentation Layer Pakete 33
Tabelle 9: Übersicht der Test Cases 52
Tabelle 10: Workflow Editor starten #TC 1.1 52
Tabelle 11: Workflow Editor beenden #TC 1.2 53
Tabelle 12: Neuer Workflow erstellen #TC 1.3 53
Tabelle 13: Bestehender Workflow öffnen #TC 1.4 54
Tabelle 14: Bestehender Workflow speichern #TC 1.5 54
Tabelle 15: Workflow schliessen #TC 1.6 55
Tabelle 16: Workflow bearbeiten #TC1.7 55
Tabelle 17: Elemente hinzufügen #TC1.8 56
Tabelle 18: Elemente entfernen #TC1.9 57
Tabelle 19: Modus & Simulation #TC 1.10 58

E. Abbildungsverzeichnis

Abbildung 1: Datenstruktur des Workflows 7
Abbildung 2: Abhängigkeiten des Workflows 8
Abbildung 3: Zustandsdiagramm Workflow Editor 9
Abbildung 4: Aktivitätsdiagramm Workflow Editor starten 11
Abbildung 5: Aktivitätsdiagramm neuer Workflow erstellen 12
Abbildung 6: Aktivitätsdiagramm Workflow laden 13
Abbildung 7: Aktivitätsdiagramm Workflow schliessen 14
Abbildung 8: Aktivitätsdiagramm Workflow speichern 15
Abbildung 9: Aktivitätsdiagramm Simulation Modus / Workflow simulieren 16
Abbildung 10: Zusammenhänge in JAXB 17
Abbildung 11:Aufbau eines Menus mit Actions & Beispiel Menu mit Sample Output 18
Abbildung 12: Austauschen eines Panels 19
Abbildung 13: Testprogramm für die Workflow Engine 20
Abbildung 14: Test Output 20
Abbildung 15: Software Layer 21
Abbildung 16: Software Layer 22
Abbildung 17: Software Layer Pakete 23
Abbildung 18: Environment und Exception Klassen 24
Abbildung 19: Data Layer Pakete 24
Abbildung 20: Data Model Klassen 25

Abbildung 21: Presentation Model Klassen 26
Abbildung 22: dto.workflow Klassen 27
Abbildung 23: dto.inventory Klassen 28
Abbildung 24: dto.catalog Klassen 29
Abbildung 25: dto.util Klassen 30
Abbildung 26: Business Layer Pakete 31
Abbildung 27: Command Handler Klasse 31
Abbildung 28: Simulation Klassen 32
Abbildung 29: Paket Diagramm Presentation Layer 33
Abbildung 30: Dialog Klassen 33
Abbildung 31: Menu und menu.action Klassen 34
Abbildung 32: View Klassen (Panels) 35
Abbildung 33: Sequenz Diagramm Startup 36
Abbildung 34: Sequenz Diagramm Init MainModel 36
Abbildung 35: Sequenz Diagramm Init MainView 37
Abbildung 36: Sequenz Diagramm Open Workflow 39
Abbildung 37: Sequenz Diagramm Create Tree 40
Abbildung 38: Sequenz Diagramm Select Element 41
Abbildung 39: Add Element (ActionGroup) 42
Abbildung 40: Erste Idee Datenfluss 43
Abbildung 41: Übersicht Datenfluss 44
Abbildung 42: Schliessen mit dem X Button 45
Abbildung 43: Ausfüllen eine Panels 46
Abbildung 44: Erster Panel Entwurf für eine Action (Aus Pflichtenheft) 49
Abbildung 45: 2-Fach Auswahlbox (Prototype) 50
Abbildung 46: Ausschnitt 3-Fach Auswahlbox (Definitive Lösung) 51
Abbildung 47: Übersicht HOVISSE Workflow Editor 61
Abbildung 48: Details zu selektiertem Element 62
Abbildung 49: Bearbeiten des Workflow Trees 62
Abbildung 50: Menu SimulationsModus 63
Abbildung 51: Menu Simulation 63
Abbildung 52: Workflow Editor während einer Simulation 63

F. Codeverzeichnis

Code 1: Schliessen mit dem X Button 46
Code 2: Ausfüllen eine Panels 46
Code 3: Beispiel hwfe.properties 60
Code 4: Beispiel log4.properties 60